Cláudio Luís Vieira Oliveira
Humberto Augusto Piovesana Zanetti

ARDUINO DESCOMPLICADO

Como Elaborar Projetos de Eletrônica

érica | Saraiva

VISIT…

LANZAROTE
Caliente.COM

Cláudio Luís Vieira Oliveira
Humberto Augusto Piovesana Zanetti

Arduino Descomplicado

Como Elaborar Projetos de Eletrônica

1ª edição

érica | Saraiva

Saraiva Educação Ltda.

Rua Henrique Schaumann, 270
Pinheiros – São Paulo – SP – CEP: 05413-010
PABX (11) 3613-3000

SAC **0800-0117875**
De 2ª a 6ª, das 8h30 às 19h30
www.editorasaraiva.com.br/contato

Gerente editorial	Rosana Arruda da Silva
Editora	Beatriz M. Carneiro
Assistentes editoriais	Paula Hercy Cardoso Craveiro Raquel F. Abranches
Produtores editoriais	Laudemir Marinho dos Santos Rosana Aparecida A. Santos
Assistentes de produção	Erika Amaro Rocha Graziele Liborni Livia Vilela de Lima Borali
Suporte editorial	Ariane Gonçalo Barboza
Produção gráfica	Liliane Cristina Gomes

Revisão	Clara Diament
Diagramação	Join Bureau
Capa	Casa de Ideias
Impressão e acabamento	Nonono

ISBN 978-85-365-1811-4

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
ANGÉLICA ILACQUA CRB-8/7057

Oliveira, Cláudio Luís Vieira
Arduino descomplicado : como elaborar projetos de eletrônica / Cláudio Luís Vieira Oliveira; Humberto Augusto Piovesana Zanetti. – São Paulo : Érica, 2015.

Bibliografia
ISBN 978-85-365-1811-4

1. Arduino (Controlador programável) 2. Linguagem de programação 3. Microcontroladores 4. Eletrônicos – Processamento de dados I. Título II. Zanetti, Humberto Augusto Piovesana

15-0027 Editado também como livro impresso CDD-005.133

Índices para catálogo sistemático:
1. Linguagem de programação

Copyright © Cláudio Luís Vieira Oliveira;
Humberto Augusto Piovesana Zanetti
2015 Saraiva Educação
Todos os direitos reservados.

1ª edição
3ª tiragem: 2016

Os Autores e a Editora acreditam que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, porventura mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site da Editora Saraiva.

A ilustração de capa e algumas imagens de miolo foram retiradas de <www.shutterstock.com>, empresa com a qual se mantém contrato ativo na data de publicação do livro. Outras foram obtidas da Coleção MasterClips/MasterPhotos© da IMSI, 100 Rowland Way, 3rd floor Novato, CA 94945, USA, e do CorelDRAW X5 e X6, Corel Gallery e Corel Corporation Samples. Corel Corporation e seus licenciadores. Todos os direitos reservados.

Todos os esforços foram feitos para creditar devidamente os detentores dos direitos das imagens utilizadas neste livro. Eventuais omissões de crédito e copyright não são intencionais e serão devidamente solucionadas nas próximas edições, bastando que seus proprietários contatem os editores.

Nenhuma parte desta publicação poderá ser reproduzida por qualquer meio ou forma sem a prévia autorização da Saraiva Educação. A violação dos direitos autorais é crime estabelecido na lei nº 9.610/98 e punido pelo artigo 184 do Código Penal.

282.024.001.003

Dedicatória

Aos meus pais, Maria Creyde e Manoel, que sempre acreditaram no "poder" da educação e dos livros.

À minha esposa Claudia e à minha filha Franciele, os grandes amores da minha vida.

À Ângela Lühmann, pelas críticas construtivas, ideias e sugestões que muito contribuíram para o desenvolvimento desta obra.

Ao bom amigo Humberto, por ter me apresentado alguns anos atrás ao universo do Arduino e pelo prazer de trabalharmos juntos nesta obra.

Cláudio Oliveira

Aos meu pais, Celia e Alberto, por todo o apoio durante toda a minha vida.

À minha esposa Flavia, por todo amor, carinho e dedicação.

Ao meu grande amigo Cláudio, por sua amizade, parceria e empenho no desenvolvimento desta obra.

À toda a comunidade Arduino!

Humberto Zanetti

Agradecimentos

Agradecemos a todos os participantes do Fatecino
(Clube de Arduino da Fatec de Jundiaí),
pois sem a existência desse clube, esta obra não existiria.

Sumário

Capítulo 1 – Arduino 17
1.1 O que é o Arduino? 17
1.2 Ambiente de desenvolvimento 18
1.3 Instalação 19
1.3.1 Instalação no Windows 7 ou 8 20
1.3.2 Instalação no MacOS 22
1.3.3 Instalação no Linux 24
1.4 Teste 25
1.5 Linguagem de programação 26
1.5.1 Estrutura básica e sintaxe do sketch 27
1.5.2 Constantes e variáveis 28
1.5.3 Tipo de dados 28
1.5.4 Operadores aritméticos, relacionais, lógicos e bit a bit 29
1.5.5 Funções 31
1.5.6 Estruturas de controle 32
Exercícios 37

Capítulo 2 – Eletrônica Básica 38
2.1 Tensão, corrente e resistência 38
2.2 Principais componentes 40
Exercícios 45

Capítulo 3 – Montagem do Primeiro Projeto 46
3.1 Identificação dos componentes 47
3.2 Montagem do circuito 48
3.3 Sketch 49
Exercícios 51

Capítulo 4 – Entradas e Saídas 52
4.1 Saída digital 52
4.2 Entrada digital 55
4.3 Saída PWM 59
4.4 Entrada analógica 62
Exercícios 74

Capítulo 5 – Porta Serial 77
5.1 Saída serial 77
5.2 Entrada serial 82
Exercícios 86

Capítulo 6 – Produzindo Som 87
6.1 Buzzer 87
6.2 Cristal piezoelétrico 90
Exercícios 93

Capítulo 7 – Displays 94
7.1 Liquid Crystal Display (LCD) 94
7.2 Displays de LED com sete segmentos 100
7.3 O circuito integrado MAX 7219 ou 7221 107
7.4 Matriz de LEDs 113
Exercícios 127

Capítulo 8 – Sensores 128
8.1 Light Dependent Resistor 128
8.2 Sensor de temperatura 131

8.3 Sensor de temperatura e umidade 141
8.4 Sensor ultrassônico 147
8.5 Sensor óptico reflexivo 154
Exercícios 157

Capítulo 9 – Módulos 159
9.1 Relógio em tempo real 159
9.2 Bluetooth 175
9.3 Controle remoto com infravermelho 180
Exercícios 185

Capítulo 10 – Motores e Servomotores 186
10.1 Motores de corrente contínua 186
10.2 Drivers e shields para motores 192
10.3 Servomotor 206
Exercícios 210

Capítulo 11 – Processing 211
11.1 Primeiros passos em Processing 211
11.2 Envio de dados do Arduino para a Processing 215
11.3 Envio de dados da Processing para o Arduino 222
11.4 Bibliotecas 227
Exercícios 243

Capítulo 12 – Android e Arduino 245
12.1 Introdução e configuração do ambiente de desenvolvimento 245
12.2 Estrutura básica de um aplicativo Android 251
12.3 Controlando um LED por um aplicativo Android 253
12.4 Matriz de LEDs controlada por Bluetooth 263
Exercícios 275

Capítulo 13 – Documentando seus Projetos com Fritzing 276
13.1 Download e instalação 276
13.2 Tutorial 277
Exercícios 281

Bibliografia **283**

Apêndice A – Outros Modelos do Arduino 285

Sobre os autores

Cláudio Luís Vieira Oliveira

É mestre em Sistemas de Computação pela Pontifícia Universidade Católica de Campinas (2005) e graduado em Análise de Sistemas pela Universidade Metodista de Piracicaba (1990). Apresenta ampla experiência na área de Ciências da Computação, com ênfase em Sistemas de Computação, atuando principalmente nos seguintes temas: sistemas de bancos de dados, linguagens de programação Java, C++, C#, programação para a web (HTML, XML, ASP, PHP, JSP e Servlets), plataforma .NET, desenvolvimento de aplicações para dispositivos móveis, redes de computadores, sistemas distribuídos, arquitetura orientada a serviços, agentes inteligentes, redes neurais artificiais e sistemas tutores inteligentes. É coordenador de curso e docente da Faculdade de Tecnologia de Jundiaí (Fatec) e docente na Faculdade de Tecnologia de Bragança Paulista (Fatec).

Humberto Augusto Piovesana Zanetti

É mestre em Ciências da Computação pela Faculdade de Campo Limpo Paulista, especialista em Administração em Sistemas de Informação pela Universidade Federal de Lavras (UFLA) e graduado em Tecnologia em Informática pela Universidade Estadual de Campinas (Unicamp). Possui dez anos de experiência em Ciências da Computação, atuando com desenvolvimento de sistemas com as linguagens de programação C, C#, Java e Python, programação para web (HTML, XML, JSP e PHP), sistemas de banco de dados, sistemas operacionais, sistemas de automação, robótica pedagógica, semiótica organizacional e tecnologias educacionais. Atualmente é professor na Escola Técnica Estadual (Etec) Rosa Perrone Scavone de Itatiba e na Faculdade de Tecnologia (Fatec) de Jundiaí.

Prefácio

Prepare-se para uma grande aventura!

As primeiras coisas que me vieram à mente quando o Professor Humberto Zanetti me convidou para escrever o prefácio deste livro foram as sensações de curiosidade, excitação e satisfação que dominaram boa parte de minha infância em Araguari, quando eu me enfurnava nos quartos cheios de quinquilharias da casa de minha avó em busca de ferramentas e equipamentos eletrônicos sem uso para que eu pudesse desmontar. Eu tinha por volta oito anos de idade e nenhum conhecimento formal sobre eletrônica.

A primeira coisa a se fazer ao encontrar um velho rádio era procurar pelos parafusos e removê-los, um a um, ali no chão do quarto. Logo, as entranhas daquela máquina começavam a aparecer e eram, aos poucos, transformadas em um monte de peças espalhadas pelo carpete. Com o tempo, depois de fazer isso com vários outros rádios, já sabia o que esperar de cada um deles, apesar de serem todos diferentes um dos outros. Um capacitor variável responsável pela sintonia, uma placa com a circuitaria de amplificação, um módulo para segurar as pilhas e um pequeno falante para emitir o som. Mas a curiosidade não parou por ali e comecei a tentar entender como cada uma daquelas peças funcionava, fuçando nas enciclopédias e nos livros velhos de meus tios e em revistas de eletrônica que comecei a comprar na banca de revistas da cidade. Em algum momento, que não me lembro direito qual, até um curso a distância de conserto de rádio e TV eu encontrei e comecei a estudar. O mesmo caminho aconteceu quando tive acesso aos meus primeiros computadores (que também foram desmontados e depois montados novamente).

Comecei então a criar minhas próprias geringonças ligando pilhas a motores, luzes e alto-falantes. Às vezes, os projetos eram apenas para entender como um ou outro componente se comportaria ao ser conectado a outro e, em outras ocasiões, eu incrementava alguns de meus brinquedos com alguma invenção. Uma miniatura de um avião de guerra ganhou um motor para que a hélice rodasse de verdade – na realidade, na esperança de que ele voasse, mas isso nunca aconteceu – e meu primeiro computador ganhou a habilidade de ligar e desligar uma lâmpada. Não me tornei um conhecedor de eletrônica ou computação, mas aprendi muitas coisas que, depois, fizeram mais sentido quando estudadas na escola e, mais tarde, na faculdade de Engenharia da Computação.

Depois das aventuras com eletrônica e computação em minha infância, a paixão pela tecnologia havia tomado de vez meu coração. Quando estava escolhendo o que gostaria de estudar, a resposta foi simples e direta: quero fazer Engenharia da Computação na Unicamp! E foi isto que aconteceu: em 1993 fui admitido naquele curso e me graduei em 1997. Foi lá que aprendi com grandes professores as bases da eletrônica e computação, e fiz meus primeiros projetos mais complexos. Posteriormente acabei me enveredando cada vez mais pelo mundo do software e da internet e me distanciei um pouco da eletrônica. Uma das razões desse distanciamento era o fato de que, para criar meus próprios projetos – coisa que faço até hoje –, era muito simples fazer projetos com software, mas não com eletrônica. Com software, tudo o que eu precisava era de um computador e todas as ferramentas para um projeto estavam a um download de distância. Mas para projetos de eletrônica, mesmo os mais simples, era preciso de um monte de ferramentas e acesso a componentes e plataformas de desenvolvimento caras, inacessíveis ou que exigiam muito trabalho antes da verdadeira diversão.

Bom, a essa altura você deve estar se perguntando: o que essa história tem a ver com este livro? É aí que entra o Arduino. Há mais ou menos cinco anos, encontrei na web um artigo sobre uma plataforma de prototipação eletrônica que era barata, bem simples de ser utilizada e com a qual você conseguia fazer seu projeto com pouquíssimos componentes em menos de uma hora. Lembrei-me das plataformas parecidas que usei na universidade e nenhuma delas agrupava todas essas qualidades. A plataforma em questão era o Arduino. Não tive dúvidas: imediatamente comecei a procurar onde poderia comprar meu primeiro kit. Dias depois comprei meu primeiro Arduino Duemilanove no site da SparkFun Electronics e fiquei esperando a encomenda chegar. Quando o pacote chegou, abri a caixa e espalhei os componentes em minha mesa. A sensação era exatamente a mesma de abrir um antigo toca-discos. Peguei a placa Arduino, conectei a meu computador, baixei o SDK, abri um dos projetos de exemplo que fazia um LED da placa piscar e cliquei no botão para enviar o programa para a placa. Segundos depois, um sorriso se abriu em meu rosto quando vi que o LED estava piscando exatamente como o programa instruía. Mudei parâmetros do programa para o LED piscar mais lentamente, cliquei no botão novamente e, em seguida, o LED da placa passou a piscar mais lentamente. Uau!!! Era verdade, era possível trabalhar com projetos eletrônicos de maneira limpa, organizada, simples e barata. Foi como ter oito anos de idade novamente e fazer seu primeiro LED acender.

Mas não é por conta das minhas memórias de infância que você deveria ler este livro. Ao lê-lo, conhecer e brincar com a plataforma Arduino, você abrirá as portas de um mundo mágico e super poderoso. O Arduino cria uma ponte entre o mundo das ideias e os da eletrônica e computação, no qual é possível transformar ideias em realidade de maneira rápida e simples por meio de protótipos eletrônicos. A prototipação rápida, por sua vez, é um convite à experimentação e exploração do que é possível de se fazer. Seja modificando projetos já existentes ou combinando outras ideias, você verá que é possível criar projetos bem complexos sem sair de sua sala. Você nunca mais olhará para um equipamento qualquer sem imaginar como ele funciona por dentro ou como ele poderia ser melhorado com suas

próprias ideias. Além disso, e não menos importante, conhecendo a história da plataforma Arduino, você perceberá o quão grande é o poder e potencial da filosofia de abertura que é parte fundamental dele.

O Arduino é um projeto aberto, em que todo o seu código-fonte, diagramas elétricos e outras informações necessárias para construí-lo foram devidamente documentadas e tornadas públicas por seus criadores. Além de publicarem o projeto, seus criadores também o associaram a licenças que permitem que essas informações sejam utilizadas para que alguém replique o projeto sem qualquer tipo de custo e que também possa modificá-lo e distribuí-lo. É o que chamamos de hardware livre – livre para ser replicado, modificado e explorado. O Arduino é o principal marco na história do hardware livre, e se não fosse isso, talvez não estivéssemos aqui falando dele.

É isso, é hora de pegar sua caixa de ferramentas e se preparar para prazerosas horas de leitura e experimentação, pois a aventura pelo mundo da eletrônica, do hardware livre, da cultura hacker e do movimento maker já vai começar. Queime alguns componentes, erre sem medo e divirta-se.

Manoel Lemos[1]

1 Manoel Lemos é engenheiro da computação pela Universidade de Campinas (Unicamp), com especialização em Computação de Alto Desempenho pela mesma instituição e em Estratégia, Inovação e Tecnologia pelo Instituto de Tecnologia de Massachusetts (MIT). Apaixonado por tecnologia e empreendedorismo, fundou várias startups, entre elas o BlogBlogs, que foi o maior indexador de blogs em língua portuguesa do mundo. Recentemente fundou o Fazedores.Com (http://fazedores.com) para promover a cultura maker no Brasil. É ainda Partner na Redpoint e.Ventures (http://rpev.com.br), um fundo de investimentos focado em startups brasileiras.

Introdução

Você algum dia já teve a vontade de desmontar algum equipamento eletrônico, apenas para ver o que tem dentro? Ou, quem sabe, para entender seu funcionamento mais a fundo ou mesmo para simplesmente ver o que tinha dentro? Ou quem sabe até melhorá-lo? Bom, se você já passou por isso, então tem um espírito "hacker". O termo hacker é utilizado para definir alguém ou alguma atitude criminosa. Mas pouca gente sabe que foi criado nos anos 1960, em torno do Massachusetts Institute of Technology (MIT), e originalmente definia pessoas que gostavam de um desafio intelectual e criativo de entender, superar limites e estender as capacidades de sistemas computacionais. Nada relacionado a crimes. Ser hacker era tirar o máximo de alguma tecnologia, não se satisfazer somente com o que era oferecido e descobrir novas aplicações e soluções com os recursos à mão. Graças a esse espírito hacker temos, por exemplo, os computadores pessoais, nascidos em uma garagem.

Em todo o mundo essa cultura hacker possui suas ramificações, como a cultura maker, que incentiva a pessoa em criar suas próprias soluções tecnológicas. Esse conceito é uma evolução da cultura Do-it-Yourself (DIY), ou o famoso "faça você mesmo". Mas como vou criar minhas próprias soluções tecnológicas sem me aprofundar em estudos avançados em eletrônica ou computação? A resposta é simples: seja curioso!

Hoje existem muitos kits de robótica bem baratos, impressoras 3D com preços próximos às convencionais e facilidade em comprar qualquer componente eletrônico que desejar. Mas como entender tudo isso sem ter que ficar horas estudando eletrônica? Simples, a resposta é Arduino! Massimo Banzi, um dos desenvolvedores e atualmente o maior representante do projeto Arduino, relatou em seu livro *Primeiros Passos com o Arduino*, que recebeu um desafio enorme: ensinar a designers conceitos de eletrônica para que eles criassem projetos automatizados e interativos. Sua primeira tentativa foi ensinar na maneira convencional e acadêmica. Frustrou-se rapidamente, vendo seus alunos entediados assistindo sua aula. Então, adotou uma abordagem mais empírica, menos teórica e mais prática. E deu certo, muito certo!

O projeto Arduino tem essa intenção: criar uma maneira descomplicada de aprender eletrônica. Uma placa Arduino deve ser vista não só como uma plataforma de prototipagem simples para iniciantes, mas também como uma maneira de você perder o medo de aprender eletrônica. A ideia principal desta placa é possibilitar que pessoas com o mínimo de curiosidade possam criar pequenos projetos e aprender com eles. Além de descobrir as funções de cada componente eletrônico de maneira simples, segura e confortável. E, principalmente, de dar a você recursos para realizar novas criações.

Essa também é a ideia deste livro: encorajar o leitor a aprender e a criar. *Arduino Descomplicado – Como Elaborar Projetos de Eletrônica* tem como objetivo fazer com que você tenha contato com conceitos de eletrônica e programação de maneira simples, mas ao mesmo tempo motivadora, executando projetos e colocando a "mão na massa". Esperamos que a cada capítulo você conheça cada vez mais a parte técnica e tenha cada vez mais ideias e vontade de aplicar os conhecimentos adquiridos. Queremos também que você deixe de ser apenas curioso, queremos que você se torne um hacker!

1 Arduino

O objetivo deste capítulo é, inicialmente, apresentar o Arduino. Em seguida, serão mostrados os passos necessários para configurar e verificar o funcionamento do ambiente de desenvolvimento nas plataformas Windows, Linux e MacOS. A linguagem de programação utilizada para programar o Arduino também será abordada.

1.1 O que é o Arduino?

O **Arduino** é uma plataforma de hardware open source, projetada sobre o microcontrolador Atmel AVR, que pode ser programado através de uma linguagem de programação similar a C/C++, permitindo a elaboração de projetos com um conhecimento mínimo ou mesmo nenhum de eletrônica. Foi criado com o objetivo de fornecer uma plataforma de fácil prototipação de projetos interativos, unindo software e hardware, características da Computação Física.

A Computação Física é uma área da Computação na qual o software se comunica diretamente com o hardware, controlando componentes eletrônicos, como sensores e atuadores, permitindo construir sistemas que consigam perceber e interagir com ambientes reais.

A placa Arduino é muito similar à de um computador de pequeno porte, sendo composto por um microcontrolador, memória RAM, armazenamento secundário (memória flash) e clock, entre outras funcionalidades. Na Figura 1.1 temos o modelo Uno, que é um dos modelos mais vendidos e bastante indicado para os iniciantes na plataforma.

Este modelo apresenta 14 pinos que podem ser utilizados como entradas ou saídas digitais (pinos 1 a 14), e os pinos 3, 5, 6, 9, 10 e 11 podem ser utilizados para gerar um conjunto de valores inteiros entre 0 e 1.023 pela técnica de Pulse Width Modulation (PWM), abordada, posteriormente, no Capítulo 4. Os pinos A0 a A5 correspondem às entradas analógicas, enquanto os 3, 3 V, 5 V e GND (Terra) permitem alimentar os componentes dos circuitos conectados ao Arduino. Possui um microprocessador ATmega328, com uma memória RAM de 2 KB, memória Flash de 32 KB e velocidade de clock de 16 MHz.



Figura 1.1 – Arduino Uno.

Mesmo tendo similaridades com computadores convencionais, a principal diferença do Arduino está na maneira como utilizamos as entradas e saídas. Por exemplo, em um computador pessoal, emprega-se teclado e mouse para entradas e monitores e impressoras para saídas. Já no Arduino, as entradas e saídas estão diretamente ligadas a componentes eletrônicos. Por isso, uma entrada do Arduino pode estar lendo o valor de um sensor de temperatura, interpretando-o e dando como resposta, por exemplo, o acendimento de um LED.

1.2 Ambiente de desenvolvimento

Assim como em qualquer dispositivo programável, a plataforma Arduino necessita que os programas sejam desenvolvidos em uma linguagem de programação, compilados e, posteriormente, transferidos para o Arduino, de modo que seja possível a execução dos comandos utilizados no programa. Com o intuito de facilitar e tornar o processo mais produtivo, utilizamos um programa denominado ambiente integrado de programação, comumente chamado de IDE, do inglês Integrated Development Environment.

Um programa criado para o Arduino é chamado de sketch, e o ambiente de desenvolvimento apresenta uma interface para o usuário bastante simples e intuitiva, apresentando recursos para abrir e salvar os sketches, transferir os programas criados para a placa, selecionar qual modelo do Arduino será utilizado, entre várias outras funcionalidades.

Nesse ambiente de desenvolvimento, ilustrado na Figura 1.2, você desenvolverá os projetos apresentados neste livro e irá transferi-los pela conexão USB para o Arduino.



Figura 1.2 – Ambiente de desenvolvimento.

1.3 Instalação

O ambiente de desenvolvimento possui versões para várias plataformas diferentes e está disponível gratuitamente para download no site oficial do Arduino, na página http://arduino.cc/en/main/software mostrada na Figura 1.3. Em seguida, faça o download do arquivo indicado para a plataforma que você utilizará.



Figura 1.3 – Página para download do Arduino IDE.

1.3.1 Instalação no Windows 7 ou 8

Depois de realizar o download do ambiente de desenvolvimento (Arduino IDE), disponível no link Windows Installer ilustrado na Figura 1.3, conecte o Arduino ao seu computador por meio do cabo USB. O sistema operacional tentará fazer o reconhecimento do novo hardware, porém falhará, como mostrado na Figura 1.4. Não se preocupe, isso é esperado: apenas clique no botão "Fechar".

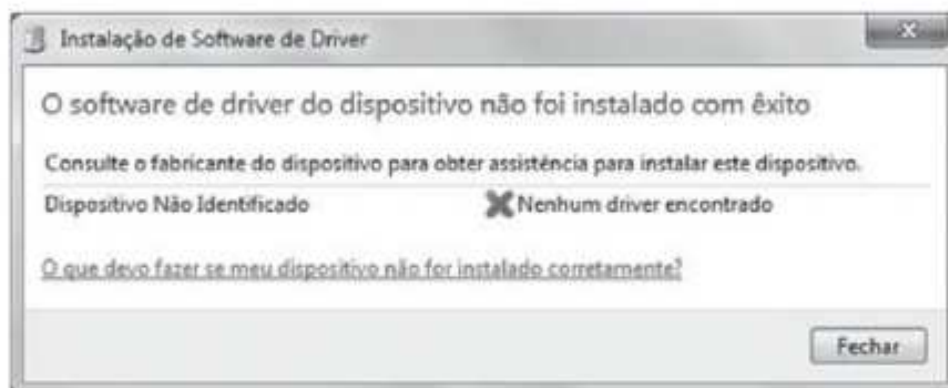


Figura 1.4 – Página para download do Arduino IDE.

Continue com a instalação e execute o arquivo "arduino-1.0.6-windows.exe" (ou versão mais recente) que foi baixado. Em seguida, pressione o botão "I Agree" para concordar com os termos da licença e prosseguir com o processo de instalação (Figura 1.5).

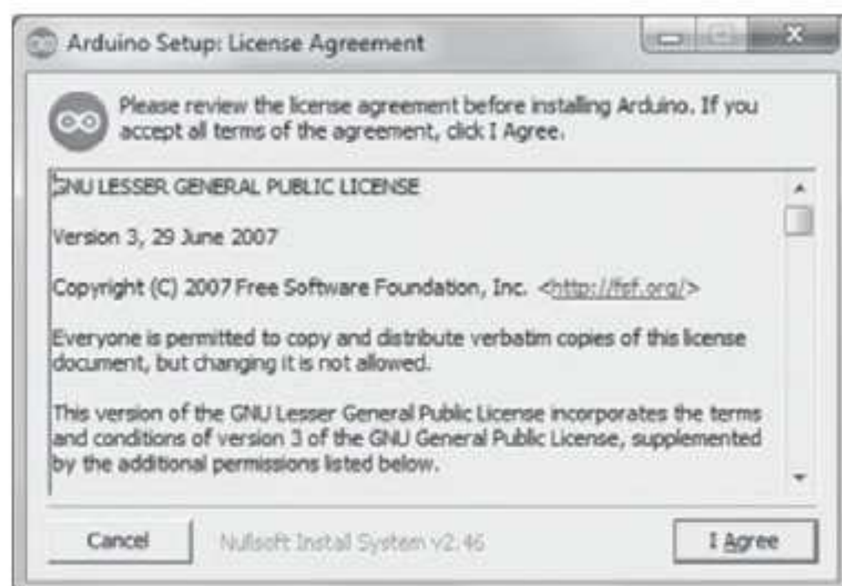


Figura 1.5 – Termos da Licença de Uso do Programa.

Na janela apresentada na Figura 1.6, mantenha os itens que estão selecionados por padrão e pressione o botão "Next >".



Figura 1.6 – Opções de instalação.

Agora, você deve selecionar a pasta na qual ocorrerá a instalação do ambiente de desenvolvimento. Em seguida, pressione o botão "Install", conforme podemos notar na Figura 1.7.

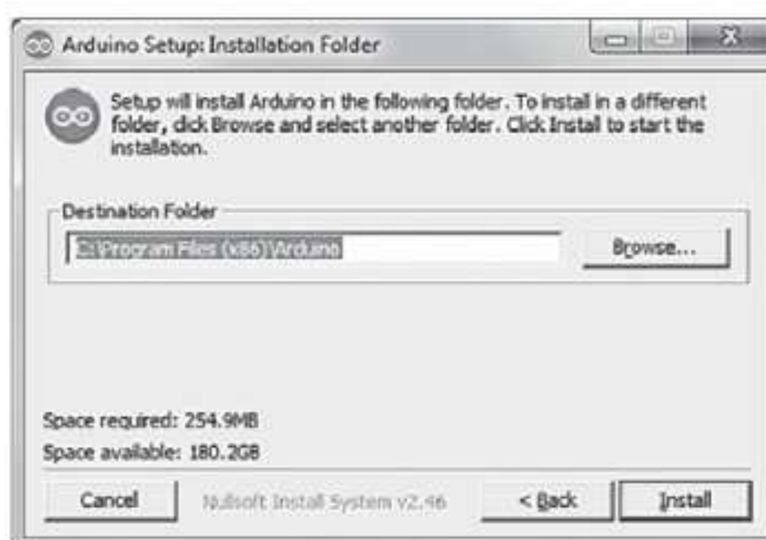


Figura 1.7 – Pasta da instalação.

Aguarde a processo de instalação e, quando aparecer a janela mostrada na Figura 1.8, que solicita a permissão do usuário para realizar a instalação do driver USB para o Arduino, clique no botão "Instalar".



Figura 1.8 – Instalação do driver USB para o Arduino.

Ao término da instalação, pressione o botão "Close" (Figura 1.9) e o ambiente de desenvolvimento estará pronto para ser utilizado.

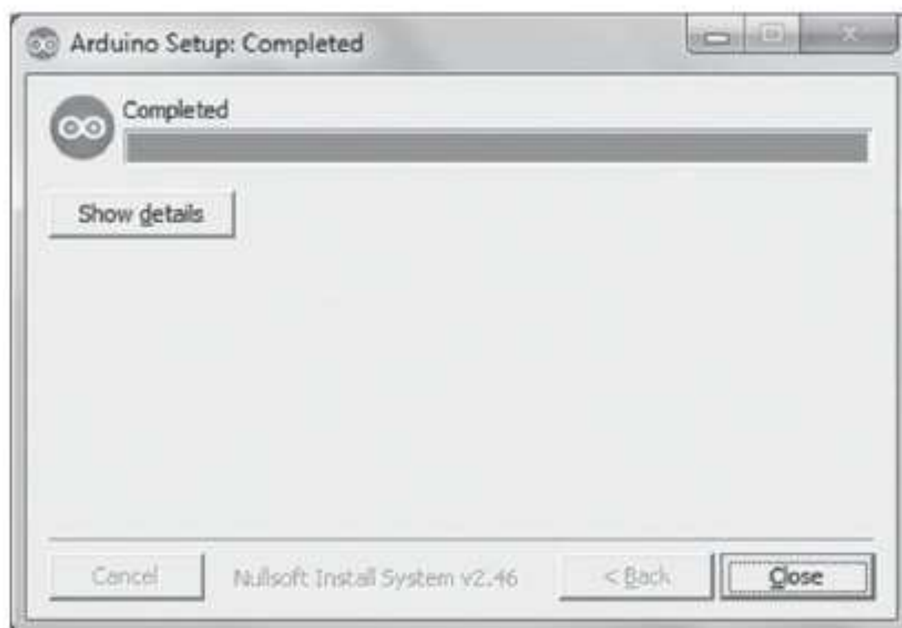


Figura 1.9 – Instalação do driver USB para o Arduino.

Após isso, basta abrir o ambiente de desenvolvimento do Arduino e realizar o teste que se encontra descrito no item 1.4.

1.3.2 Instalação no MacOS

Após realizar o download da última versão mais recente do ambiente de desenvolvimento pelo link "Mac OS X", disponível no endereço http://www.arduino.cc/en/Main/Software, um arquivo com a extensão ".zip" será baixado no computador. Em seguida, basta descompactar o aplicativo do Arduino, por meio de um duplo clique sobre o ícone do arquivo.

Quando utilizar as as placas Arduino Uno e Mega, não é necessário baixar e instalar nenhum drive adicional. Porém, caso você esteja utilizando alguma placa com chip FTDI, como, por exemplo, os modelos Duemilanove, Diecimila e Nano, torna-se necessário realizar o download de um instalador para o grupo de drives que está disponível no endereço http://www.ftdichip.com/Drivers/VCP.htm. Após realizar o download do instalador, dê um duplo clique no arquivo e siga as instruções do instalador. Ao final do processo, é necessário reiniciar o computador.

Após descompactar o aplicativo Arduino, dê um duplo clique no ícone e a janela mostrada na Figura 1.10 será aberta.



Figura 1.10 – Arduino IDE.

Em seguida, no menu do ambiente de desenvolvimento do Arduino, escolha "Tools", depois "Boards", e selecione o modelo da sua placa, conforme podemos notar na Figura 1.11.

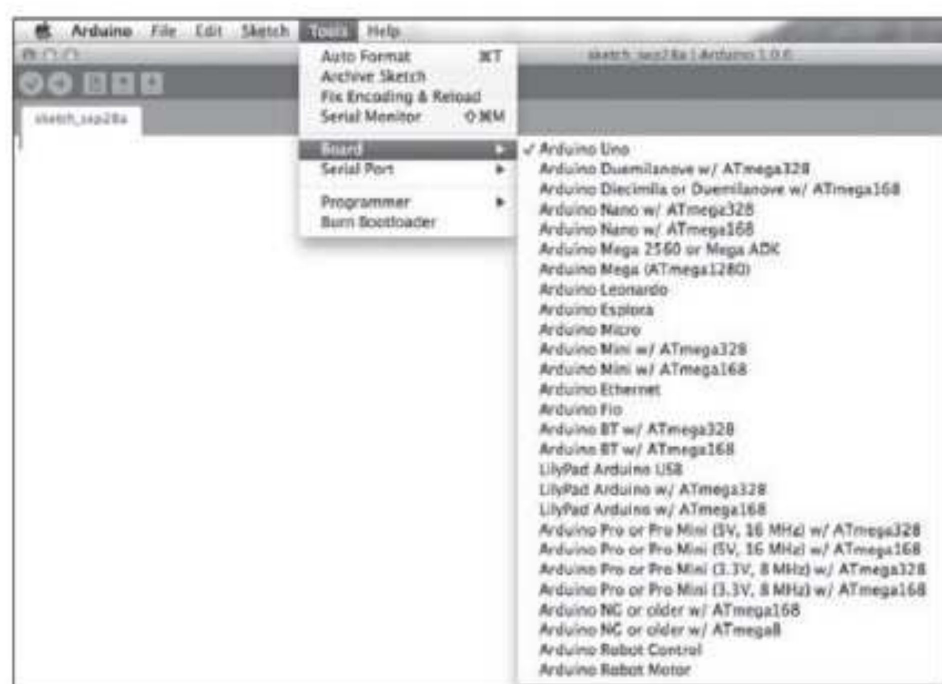


Figura 1.11 – Escolha da placa do Arduino.

Na Figura 1.12 podemos observar a escolha da porta serial. Para isso, acesse novamente o menu, escolha "Tools", "Serial Port", e selecione o dispositivo serial /dev/tty.usbmodem ou /dev/cu.usbmodem para placas Uno ou Mega. Caso sua placa seja com o chip FTDI (Duemilanove, Diecimila e Nano), selecione a opção /dev/tty.usbserial ou /dev/cu.usbserial.

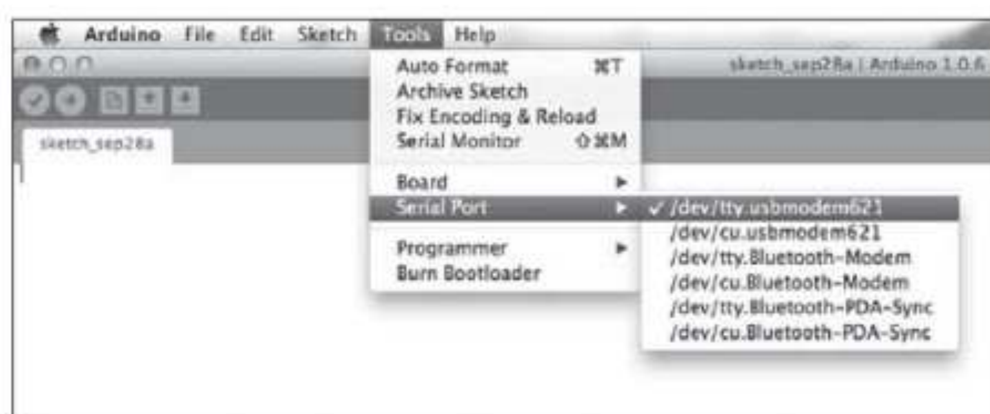


Figura 1.12 – Seleção da porta serial.

Em seguida, o Arduino IDE está instalado. Passe para o item 1.4 e realize o teste que se encontra descrito ali.

1.3.3 Instalação no Linux

Este tópico aborda a instalação na distribuição Linux Ubuntu 10.10 (Maverick), em similares ou mais recentes. Se você deseja realizar a instalação para outras distribuições ou versões do Linux, acesse o endereço http://playground.arduino.cc/Learning/Linux.

As janelas mostradas nas Figuras 1.13 e 1.14 referem-se especificamente à instalação do Lubuntu 14.04, porém o procedimento é idêntico para qualquer outra distribuição ou versão do Linux Ubuntu a partir do 10.10, pois há um pacote do Arduino IDE disponível no repositório "universe". Dessa forma, utilize a "Central de Programas do Ubuntu" (Figura 1.13) ou qualquer outro gerenciador de pacotes, como, por exemplo, o synaptic ou o apt-get, para realizar a instalação.



Figura 1.13 – Central de Programas do Ubuntu.

Dessa maneira, o processo de instalação será totalmente automatizado e todas as dependências de pacotes serão automaticamente instaladas. Então, apenas clique no botão "Instalar Pacotes", conforme ilustra a Figura 1.14.



Figura 1.14 – Instalação do pacote e dependências.

Após realizar o processo de instalação, vá para o item 1.4 e execute o teste que se encontra descrito ali para certificar-se de que o ambiente de desenvolvimento reconhece a placa do Arduino que estará conectada em uma das portas USB do computador.

1.4 Teste

Após instalar o ambiente de desenvolvimento e o driver USB para o Arduino, realize um pequeno teste para verificar se a instalação foi bem-sucedida.



Programa

Abra o ambiente de desenvolvimento e digite o programa a seguir.

```
int LED = 13;                  // Pino no qual o LED está conectado

void setup() {
  pinMode(LED, OUTPUT);        // Definir o pino como saída
}

void loop() {
  digitalWrite(LED, HIGH);     // Acender o LED (Nível 1 no pino)
  delay(2000);                         // Aguardar por 2 segundos
  digitalWrite(LED, LOW);      // Nível 0 no pino, apagar o led
  delay(2000);
}
```

O Arduino possui um LED na própria placa conectado ao pino 13. Dessa forma, sem montar um circuito e apenas conectando o Arduino à porta USB do computador, é possível carregar um programa, chamado de sketch, e verificar se a instalação está funcionando corretamente. Após digitar o programa, pressione o botão "Carregar", conforme indicado pela seta na Figura 1.15.



Figura 1.15 – Sketch para verificação da instalação.

Após alguns segundos, o programa será carregado na memória flash do Arduino e o LED 13, presente na placa, começará a piscar em um intervalo de dois segundos. Em caso de algum problema, verifique no menu "Ferramentas" se o modelo da placa está correto e também se a porta serial foi selecionada corretamente.

1.5 Linguagem de programação

O Arduino, assim como qualquer outro dispositivo programável, necessita de uma linguagem de programação. Podemos entender, de uma maneira sintética, que uma linguagem de programação é um conjunto de comandos que permitirão o desenvolvimento de um programa para computador ou qualquer outro dispositivo programático. Esses comandos possuem regras que podem ser classificadas como sintáticas e semânticas. A sintaxe de uma linguagem corresponde ao conjunto de palavras que podem ser usadas para expressar os comandos. Já a semântica diz respeito ao significado dessas palavras (comandos), indicando quais ações e funções devem ser realizadas. A solução para um determinado problema computacional, pensada pelo programador e descrita em linguagem de programação, é chamada algoritmo.

Por intermédio da linguagem, o programador consegue especificar como os dados serão tratados, armazenados e manipulados em um computador. Essa organização de comandos e regras é chamada de código-fonte. Uma vez criado esse código-fonte, temos toda a solução lógica (algoritmo) e comandos. Este é traduzido para uma linguagem de mais baixo nível, de máquina, para ser compreendida e executada pelo processador.

No Arduino utilizamos o ambiente de desenvolvimento integrado (IDE). O nosso código-fonte é chamado sketch, e a linguagem de programação para o Arduino é chamada Wiring. A interface gráfica desse ambiente de programação é desenvolvida em Java, baseada em um projeto chamado Processing, que será abordado em mais detalhes no Capítulo 11 desta obra. Quando escrevemos algum código-fonte e verificamos sua sintaxe e sua execução, utilizamos um compilador, cuja função é traduzir nosso código-fonte para outra linguagem, compreendida pelo microprocessador ou microcontrolador. Após o processo de compilação, o sketch poderá ser transferido, por meio de uma conexão USB, para o Arduino.

1.5.1 Estrutura básica e sintaxe do sketch

A estrutura básica de um sketch escrito em Wiring é formada por duas funções principais:

- **void setup():** utilizada para a inicialização ou configurações iniciais, daí o nome setup, do programa no Arduino. Geralmente definimos quais são as configurações iniciais, como quais pinos serão usados e qual será o modo de uso (INPUT ou OUTPUT). Os comandos inseridos nesta função são executados somente uma vez logo no início do programa.
- **void loop():** executada indefinidamente enquanto a placa estiver ligada. Seu comportamento padrão é de um ciclo de repetição, daí o nome loop. Dessa maneira, quando a última linha de comando for executada, o sketch será lido novamente a partir da primeira, e assim sucessivamente, até que a placa seja desligada ou o botão reset seja pressionado.

Toda linguagem de programação possui um conjunto básico de elementos (símbolos e palavras) de sintaxe para a escrita de comandos. Entre os mais comuns que serão apresentadas neste livro temos:

- O ponto e vírgula (;) é obrigatório para marcar o final de uma linha de comando.
- As chaves ({ }) indicam um bloco de comandos, muito comum na definição de funções e estruturas de comando.
- Os símbolos // ou /* */ denotam comentários, ou seja, trechos do programa que não são considerados pelo compilador, isto é, são ignorados e não são executados. A utilização de duas barras consecutivas indica uma linha de comentário. Se quisermos demarcar mais de uma linha, utilizamos no início da primeira o símbolo /*, abrindo o bloco de linhas de comentários e, no final da última linha o símbolo */, para fechar esse bloco de comentários. Comentários são muito úteis para que o programador documente e coloque informações relevantes sobre o código-fonte, facilitando a análise e o entendimento do programa.
- A diretiva #include é usada para incluirmos uma biblioteca de comandos. O uso e os exemplos de bibliotecas serão usados posteriormente neste livro.

1.5.2 Constantes e variáveis

Constantes são valores predefinidos e que não podem ser alterados. A linguagem de programação usada no Arduino apresenta três grupos básicos de constantes que podem ser utilizadas diretamente:

- **HIGH e LOW:** são valores referentes à tensão nos pinos digitais. HIGH indica 5 volts, e LOW indica 0 volt.
- **INPUT e OUTPUT:** são valores que definem o estado de uso de um pino. INPUT indica entrada, como quando usamos um sensor que fornece um valor de entrada (input) para a placa. O estado OUTPUT determina que o pino da placa fornecerá uma saída de valor quando, por exemplo, acendemos um LED, dando como saída o valor HIGH (5 volts).
- **TRUE e FALSE:** são referências para valores lógicos, ou seja, verdadeiro e falso, respectivamente.

Variáveis são referências de valores que são armazenados em memória, para serem manipuladas. Para utilizar uma, é necessário declará-la. Uma declaração consiste em definir um nome (identificador) para ela e informar qual é o tipo de valor que ela armazenará. Uma declaração típica em Arduino possui este formato:

```
tipo nome = valor inicial;
```

em que temos:

- **tipo:** tipo do valor ou de dado que a variável irá armazenar, por exemplo, se será um valor numérico, uma letra ou uma palavra, entre outros tipos de dados;
- **nome:** define o nome que iremos utilizar para referenciar (usar) essa variável no decorrer do programa. Os nomes de variáveis podem conter números e letras maiúsculas ou minúsculas. Deve sempre começar com uma letra, e não pode haver espaços na formação do nome. O único caractere especial permitido é o "_" (sublinhado);
- **valor inicial:** é um parâmetro opcional, pois uma variável pode ou não receber um valor inicial ao ser criada. Esse valor é atribuído pelo sinal "=".

1.5.3 Tipo de dados

Uma linguagem de programação apresenta um grupo de tipo de dados que definem o tipo do valor a ser manipulado. Por exemplo, se precisamos armazenar a idade de uma pessoa, criamos uma variável com o nome idade e definimos que ela será do tipo numérico inteiro (ou simplesmente inteiro), já que o valor referente à idade possui essa característica. No Arduino, os tipos de dados mais utilizados estão detalhados no quadro a seguir.

Tipo	Especificação
boolean	Dados do tipo booleano podem possuir apenas o valor Verdadeiro (TRUE) ou Falso (FALSE).
byte	Um dado do tipo byte armazena um número de 8 bits sem sinal que deve possuir um valor entre 0 e 255.
char	O tipo caractere utiliza 1 byte de memória e armazena o valor de um caractere. A representação simbólica do caractere deve ser escrita entre aspas simples (' ').
int	O tipo de dados inteiro é referente aos valores conjuntos dos numéricos inteiros naturais positivos e negativos, incluindo o zero e abrangendo a faixa de -32.768 a 32.767. Necessita de 2 bytes da memória para armazenamento.
float	Tipo de dado que representa o conjunto de números reais, positivos e negativos. Chamados de números de ponto flutuante, abrangem a faixa de 3,4028235E+38 a -3,4028235E+38. São necessários 4 bytes da memória para armazenar um valor desse tipo de dados.
String	Strings representam um conjunto ou cadeia de caracteres, como quando formamos uma palavra ou frase. Seu armazenamento é variável, dependendo da quantidade de caracteres que formam a cadeia. Um valor String deve ser delimitado por aspas duplas (" ").

1.5.4 Operadores aritméticos, relacionais, lógicos e bit a bit

A partir dos valores armazenados nas variáveis é possível realizar operações aritméticas e utilizá-las em expressões por meio do emprego dos operadores relacionais e lógicos. Cada um desses grupos de operadores possui um conjunto de símbolos e respectivos significados. No quadro a seguir são listados os operadores aritiméticos utilizados na linguagem de programação adotada na construção de programas para o Arduino:

Operadores Aritméticos	
Símbolo	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
=	Atribuição
++	Incremento; dessa forma, a++ significa o mesmo que a = a + 1
--	Decremento; dessa forma, a-- representa o mesmo que a = a - 1
+=	Operação composta de adição; assim, a += b significa o mesmo que a = a + b
-=	Operação composta de subtração; assim, a -= b tem o mesmo significado que a = a - b
*=	Operação composta de multiplicação; assim, a *= b representa o mesmo que a = a * b
/=	Operação composta de divisão; assim, a /= b tem o mesmo efeito que a = a / b

Dessa maneira, poderíamos escrever da seguinte forma a soma de duas variáveis inteiras:

```
int a = 10;
int b = 5;
int c;

c = a + b;
```

em que, inicialmente, as variáveis são declaradas, por exemplo int a = 10 e, posteriormente, a e b são somadas e o resultado é armazenado na variável c.

Os operadores relacionais permitem a comparação entre valores. O resultado de uma determinada comparação sempre resultará em um valor lógico (booleano), ou seja, Verdadeiro (TRUE) ou Falso (FALSE).

Operadores Relacionais	
Símbolo	Operação
==	Igual
!=	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

Dessa forma, analisando o pequeno trecho de programa a seguir, temos que o valor armazenado na variável booleana result será TRUE, poisa variável a é maior ou igual a b.

```
int a = 10;
int b = 5;
boolean result = (a >= b);
```

Os operadores lógicos permitem conectar variáveis ou expressões, sendo o resultado final de uma expressão sempre um valor lógico (booleano), ou seja, Verdadeiro (TRUE) ou Falso (FALSE).

Operadores Lógicos	
Símbolo	Operação
&&	E (AND)
||	OU (OR)
!	NÃO (NOT)

Conforme podemos concluir após analisar o trecho de programa a seguir, o valor da variável result, neste exemplo, será FALSE. Ou seja, a variável x é maior ou igual a 10, resultando em TRUE, porém a variável x não é menor que 20, resultando em FALSE. Quando usamos o operador lógico E (&&) para concatenar as duas expressões, teremos (TRUE && FALSE), o que produzirá um resultado FALSE. Pois, em uma expressão com E (AND), a saída (resultado) será verdadeiro se, e apenas se, todos os elementos da expressão (entrada) forem verdadeiros.

```
int x = 30;
boolean result = ((x >= 10) && (x < 20));
```

Os operadores bit a bit permitem a manipulação dos bits que compõem determinado dado armazenado.

Operadores Bit a Bit	
Símbolo	Operação
&	E (AND)
|	OU (OR)
~	NÃO (NOT)
^	OU Exclusivo (XOR)
<<	Deslocamento de bit à esquerda
>>	Deslocamento de bit à direita

Por exemplo, no trecho de programa seguir, a variável inteira x está armazenando o valor inteiro dois, cuja representação binária é 0000000000000010. O valor da variável y será obtido realizando o deslocamento à esquerda de três bits da variável x, ou seja, em binário, teremos 0000000000010000, que, em decimal, é o valor 16.

```
int x = 2;     // Representação binária: 0000000000000010
int y = x<< 3; // Representação binária: 0000000000010000
```

1.5.5 Funções

A utilização de funções em linguagens de programação é muito comum e traz como principal vantagem soluções para determinados problemas já desenvolvidas, que podem facilitar muito o desenvolvimento dos programas. Como exemplo, vamos citar a função

digitalWrite, que realiza o envio de um sinal digital para um determinado pino do Arduino. Geralmente, funções precisam de parâmetros (valores) que são colocados entre os parênteses localizados logo após o nome da função. No trecho de programa a seguir temos um exemplo de uso da função digitalWrite.

```
int LED = 13;
digitalWrite(LED, HIGH);
```

Podemos observar que a função digitalWrite recebe dois valores como parâmetros. O primeiro consiste em uma variável inteira, neste exemplo chamada de LED, que indica o pino para o qual o sinal digital deverá ser enviado. Também podemos notar que o segundo parâmetro consiste no valor que será enviado, neste exemplo a constante chamada HIGH.

A linguagem de programação define e fornece inúmeras funções para grupos de diferentes soluções e recursos. Nos capítulos que virão, usaremos diversas dessas funções e serão dados os detalhes de seu uso. A seguir temos alguns exemplos de funções bastante utilizadas no desenvolvimento de sketches para o Arduino:

- **define o modo de uso do pino:** pinMode;
- **uso da entrada e saída digital:** digitalWrite e digitalRead;
- **utilização da entrada e saída PWM:** analogWrite e analogRead;
- **funções de tempo:** millis e delay;
- **acesso à comunicação serial:** Serial.begin, Serial.read e Serial.print.

1.5.6 Estruturas de controle

Normalmente, não é possível desenvolver soluções para problemas computacionais somente por meio de comandos que são executados de maneira exclusivamente sequencial. Muitas vezes, precisamos agrupar comandos que só serão feitos quando uma condição ocorrer, ou de maneira cíclica (repetitiva), dependendo das características do problema. Para isso, usamos estruturas de controle, que podem ser classificadas como estruturas de decisão e de repetição.

Estruturas de decisão agrupam um grupo de comandos (blocos) que podem ser executados ou não, dependendo do resultado de uma expressão lógica. A estrutura de decisão mais simples que temos no Arduino é a estrutura *if* (em português "se"). Nela, definimos um bloco de comandos, que estão definidos entre as chaves "{ }". Esse bloco será executado somente se a expressão resultar em um valor TRUE (verdadeiro). Caso contrário, não será executado comando algum. Podemos entender essa estrutura como sendo: "se a expressão avaliada resultar em verdadeiro, o bloco de comando será executado". Assim, temos a seguir a definição da estrutura de seleção if.

```
//Se expressao igual a TRUE, o bloco de comandos é executado

if (expressao) {
  [bloco de comandos]
}
```

No exemplo mostrado no trecho de sketch a seguir, podemos observar que o bloco de comandos será executado apenas se a variável x possuir um valor menor ou igual a 10. Dessa maneira, quando essa expressão resultar em TRUE, o valor da variável x será adicionado a 10, resultando em x igual a 15. Em seguida, na próxima linha do bloco de repetição, a variável y, que vale inicialmente um, será acumulada, ou seja, ocorrerá a adição de um ao valor de x que é 15, resultando em y igual a 16.

```
int x = 5;
int y = 1;
if (x <= 10) {
  x = x + 10;
  y += x;
}
```

Podemos ter situações em que necessitamos que um bloco de comando seja executado caso a expressão lógica da estrutura if resulte em um valor FALSE. Para isso, adicionamos uma estrutura ao if, chamada else, ou seja, senão (caso contrário). Dessa forma, a estrutura else demarca um bloco de comandos que será executado quando a expressão em if seja avaliada como FALSE. Quando a expressão for FALSE e houver uma estrutura else, o bloco demarcado pelo if não será executado e apenas o bloco demarcado pelo else será executado.

É importante notar que um else necessita obrigatoriamente de uma estrutura anterior if. Então, podemos entender essa estrutura if-else como: "se a expressão da estrutura if resultar TRUE, o bloco de comando 1 será executado, senão o bloco de comandos 2 será executado", ou seja:

```
// Se expressão igual a TRUE, o  bloco de comandos-1 é executado,
// caso contrário, ou seja a expressão igual a FALSE, o
// bloco de comandos-2 será executado

if (expressao) {
  [bloco de comandos-1]
}
else {
  [bloco de comandos-2]
}
```

Podemos encadear várias estruturas if e else em situações em que se torna necessário verificar duas ou mais expressões. A seguir, podemos observar como realizar o encadeamento para um problema que exige a avaliação de duas expressões.

```
// Se expressão-1 igual a TRUE, o bloco de comandos-1 é executado.
// Caso a expressão-1 seja FALSE e a expressão-2 for TRUE o bloco
// de comandos-2 será executado. Finalmente, se a expressao-2 for
// FALSE, o bloco de comandos-3 é executado.

if (expressão-1) {
  [bloco de comandos-1]
}
else if(expressão-2){
  [bloco de comandos-2]
}
else {
  [bloco de comandos-3]
}
```

Quando temos que criar um encadeamento a partir de uma sequência de comparações com uma mesma variável para verificarmos seu valor entre vários valores possíveis, podemos utilizar a estrutura switch-case, ou seja, escolha-caso. Nessa estrutura definimos inicialmente qual é a variável a ser comparada, colocando entre os parênteses da estrutura switch. Em cada um dos termos case, colocamos o valor a ser comparado. Essa estrutura é similar ao uso de if-else encadeados com a mesma variável sendo comparada com expressões de igualdade, por exemplo quando queremos comparar se o valor de uma variável do tipo inteira é igual 1, ou se é igual a 2, ou se é igual a 3. A variável definida é comparada com todos os valores nos termos case, e quando uma das operações é TRUE, o bloco de comandos correspondente é executado. A sintaxe da estrutura switch-case é:

```
// Caso o valor da variavel seja igual a valor1, o bloco de
// comandos-1 é executado. Caso o valor da variavel seja igual
// a valor2, o bloco de comandos-2 é executado e
// assim sucessivamente. Quando todos os cases forem FALSE o
// bloco de comandos default é executado.

switch (variavel) {
  case valor1:
    [bloco de comandos-1]
    break;
  case valor2:
    [bloco de comandos-2]
    break;

  ...
```

```
  case valorN:
    [bloco de comandos-N]
    break;
  default:
    [bloco de comandos-default]
  break;
}
```

Ao final de cada bloco case, usamos o comando break, que faz com que a estrutura switch-case seja encerrada. Ao final da estrutura podemos usar um bloco default, que funciona como um "caso contrário", ou seja, executa um bloco de comandos caso nenhuma das comparações nos termos case seja TRUE. O bloco default não é obrigatório na estrutura switch-case.

A seguir, podemos observar um trecho de programa que ilustra o uso do switch-case.

```
int x = 1;
String extenso;
switch (x) {
  case 1:
    extenso = "Um";
    break;
  case 2:
    extenso = "Dois";
    break;
  default:
    extenso = "Não sei!";
    break;
}
```

Várias das soluções em programação necessitam que grupos de comandos sejam executados repetidas vezes. Para isso, precisamos criar um laço de repetição (ou loop). As estruturas de repetição utilizam uma condição no formato de expressão lógica. Enquanto essa expressão for TRUE, o ciclo de repetição continua; no momento em que a expressão se torna FALSE, o laço de repetição se encerrará.

Uma das estruturas de repetição que temos é o while, ou seja, "enquanto". Essa estrutura também utiliza { } para demarcar o bloco de comandos que deverá ser repetido. Podemos entender essa estrutura como sendo: "enquanto a expressão for TRUE, repita os comandos". A sintaxe da estrutura while pode ser vista a seguir.

```
// Enquanto expressão igual a TRUE, o bloco de
// comandos é executado

while (expressao) {
  [bloco de comandos]
}
```

No trecho de programa a seguir temos um exemplo do bloco de repetição. Nesse exemplo, enquanto o valor da variável x for menor que dez, o seu valor será incrementado (x++) e o laço de repetição se encerrará quando x atingir o valor 10, pois a expressão usada na estrutura while será FALSE.

```
int x = 0;
while (x < 10) {
  x++;
}
```

Uma estrutura de repetição muito parecida com while é o do-while, ou seja "faça-enquanto". A única diferença entre elas é o momento no qual a expressão é verificada. No while, a expressão é verificada sempre antes da execução do ciclo de repetição para determinar se deve ou não ocorrer. Por outro lado, na estrutura do-while, a expressão é verificada ao final de cada ciclo da repetição, de forma a verificar se o próximo ciclo será executado. Na prática, com do-while, o bloco será sempre executado ao menos uma vez, antes de a expressão ser checada. A sintaxe da estrutura do-while é apresentada a seguir.

```
// Executa o bloco de comandos e, em seguida, verifica se a expressão é TRUE
para executar novamente o bloco de comandos. Caso a expressão resultar em FALSE
o ciclo é encerrado.

do {
  [bloco de comandos]
} while (expressao);
```

No trecho de programa a seguir podemos observar um exemplo de utilização dessa estrutura de repetição.

```
int x = 0;
do {
  x++;
} while (x< 10);
```

As estruturas de repetição, descritas anteriormente, dependem de que a expressão, em algum momento, se torne FALSE para que o laço seja encerrado. Essa condição normalmente pode depender de alguma mudança de valor em uma variável em decorrência de algum evento, por exemplo, a mudança de valor de um sensor conectado ao Arduino. Mas algumas vezes queremos ter o controle da quantidade de vezes em que o laço de repetição ocorrerá. Para isso, precisamos controlar quando houve a repetição do bloco de comandos. Para realizar essa tarefa temos a estrutura de repetição for, ou, em português,

"para". Diferentemente das estruturas de repetição anteriores, o for precisa de outros dois parâmetros para funcionar, além da expressão, conforme podemos observar a seguir.

```
// Enquanto expressão igual a TRUE, o bloco de comandos
// é executado

for (contador; expressao; incremento-do-contador) {
  [bloco de comandos]
}
```

A variável contador é do tipo inteiro e é normalmente inicializada com 0 ou 1. O parâmetro expressão determina qual é a contagem máxima a ser feita. O parâmetro incremento informa qual valor será incrementado na variável contador a cada repetição. É importante notar que os parâmetros devem estar separados por ponto e vírgula (;). Por exemplo, se quisermos criar um laço de repetição que se repita 10 vezes, a estrutura for ficaria assim:

```
int y = 0;
for(intx = 1; x<= 10; x++) {
  y = y + x;
}
```

Observe, em tal exemplo, que inicialmente declaramos a variável x e inicializamos o seu valor com 1, em seguida a expressão (x < = 10) determinará que a repetição ocorrerá enquanto essa expressão for TRUE. Após a execução do bloco de comandos, ocorrerá o incremento da variável (x++).

Exercícios

1. Pesquise e liste quais são as principais diferenças entre os modelos Uno, Mega e Leonardo.
2. Busque algumas fontes de referência na internet e pesquise sobre a origem da linguagem de programação usada no Arduino (Wiring).
3. Explore um pouco mais seu Arduino. Localize em sua placa o botão "Reset" e descreva qual é a sua função.

2 Eletrônica Básica

Uma das vantagens e grande apelo do Arduino é a possibilidade de se desenvolver projetos de Eletrônica com pouco ou quase nenhum conhecimento de eletrônica. Porém, algumas informações essenciais que envolvem os conceitos de tensão, corrente e resistência (Lei de Ohm), e também o princípio de funcionamento e a identificação correta dos principais componentes eletrônicos, são fundamentais para o desenvolvimento dos projetos. Dessa maneira, neste capítulo, temos uma abordagem bastante acessível a todos esses conceitos.

2.1 Tensão, corrente e resistência

A eletrônica está fundamentada sobre os conceitos de tensão, corrente e resistência. Podemos entender como tensão a energia potencial armazenada em uma pilha ou bateria e que fluirá quando um circuito for fechado, através de um meio condutor, entre os polos de maior e menor potenciais (sentido convencional). Como analogia, podemos considerar a água armazenada em dois recipientes conectados por um cano (meio condutor). A água fluirá do recipiente que possui maior quantidade de água para o menor, ou seja, considerando a Figura 2.1, a água fluirá do Recipiente A para o B.

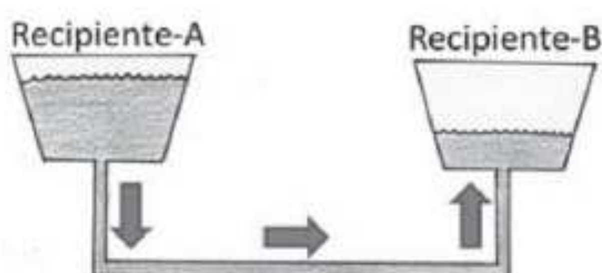


Figura 2.1 – Diferença de potencial.

Em eletrônica, o princípio é o mesmo. Por exemplo, os polos positivos e negativos de uma pilha, ou qualquer outra fonte de energia, indicam o sentido no qual a corrente elétrica fluirá. Dessa forma, podemos definir que a corrente elétrica é a movimentação ordenada de cargas elétricas através de um meio condutor. Para efeito de análise dos circuitos elétricos e eletrônicos, podemos observar na Figura 2.2 que a corrente elétrica poderá

circular em dois sentidos: a) **sentido real**, é chamado assim pois, em termos de física, é o que realmente ocorre em um circuito elétrico, isto é, o movimento de cargas negativas do menor para o maior potencial; ou b) **sentido convencional,** que resulta do movimento das cargas positivas, ou seja, do polo de maior potencial para o de menor. O sentido convencional é mais utilizado para fins didáticos, com o intuito de facilitar o entendimento de alguns conceitos.



Figura 2.2 – Sentido da corrente elétrica.

Existem dois tipos de circuitos eletrônicos. Aqueles fundamentados sobre uma fonte de energia de **corrente contínua** (Figura 2.3), ou seja, pilhas, baterias ou qualquer outro elemento que apresenta um polo positivo (+) e outro negativo (–).

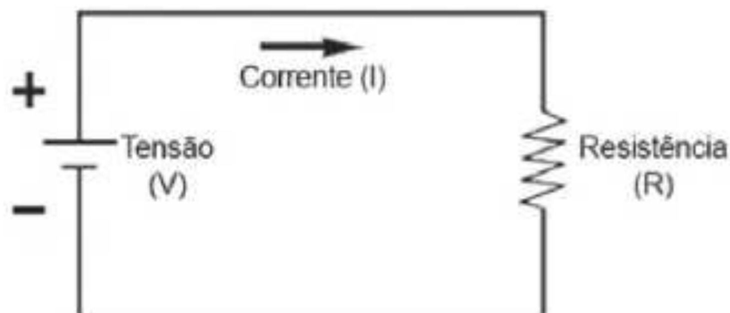


Figura 2.3 – Circuito de corrente contínua.

Também existem os circuitos de **corrente alternada** (Figura 2.4), dos quais podemos citar como melhor exemplo as tomadas residenciais, que fornecem uma alimentação de 110 ou 220 volts.

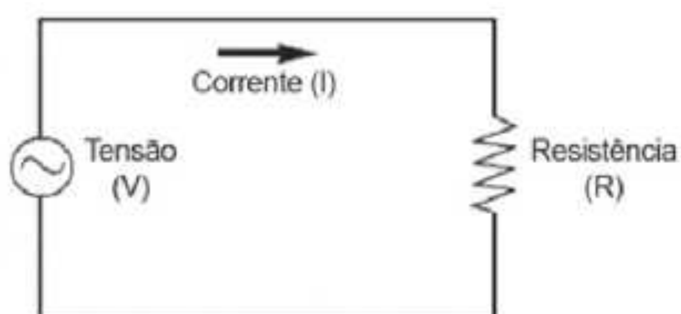


Figura 2.4 – Circuito de corrente alternada.

A movimentação das cargas elétricas através de um meio condutor pode encontrar elementos que oferecem certa resistência à passagem dessas cargas. Na Figura 2.4, por exemplo, a resistência poderia ser o filamento de uma lâmpada: a passagem da corrente elétrica produz aquecimento e o filamento fica incandescente. É esse mesmo efeito que permite que a água de um chuveiro seja aquecida ao passar pela resistência.

A Lei de Ohm estabelece a relação entre tensão (V), corrente (I) e resistência (R), em que:

$$I = V/R.$$

A tensão é expressa em volts (V); a corrente, em amperes (A); e a resistência, em ohm (Ω). Dessa forma, o circuito elétrico ilustrado na Figura 2.5, no qual temos uma tensão de 5 V aplicada sobre uma resistência de 220 Ω, produzirá uma corrente de 0,022 A ou 22 mA.

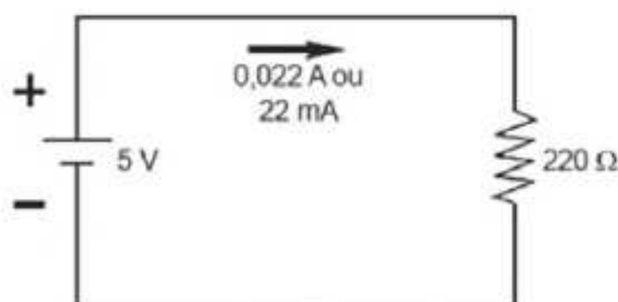


Figura 2.5 – Tensão, corrente e resistência.

2.2 Principais componentes

Um circuito eletrônico é formado por diversos componentes que apresentam finalidades distintas. Vamos a seguir aprender a identificar os mais utilizados.

Os **resistores**, conforme já explicado anteriormente, têm a função de limitar a corrente elétrica. Eles são necessários de modo a evitar que determinados componentes eletrônicos recebam uma tensão ou corrente maior do que podem suportar, evitando, dessa forma, que sejam danificados. Possuem dois terminais para conexão e são componentes não polarizados, ou seja, podem ser instalados em qualquer sentido no circuito elétrico, sem preocupação com os polos negativos ou positivos.



Figura 2.6 – Resistor (para versão colorida consulte o site da Editora Érica).

Conforme podemos observar na Figura 2.6, o valor de um resistor pode ser determinado por meio de uma tabela de código de cores. Para identificar o valor do resistor, posicione-o com a faixa dourada ou prata do lado direito e, em seguida, faça a leitura das faixas a partir da esquerda. Vamos tomar como exemplo a Figura 2.6, que apresenta um resistor com as faixas amarela, violeta, laranja e dourada. Dessa forma, consultando a tabela a seguir, temos que a primeira faixa (amarela) possui o valor 4, a segunda faixa (violeta) possui

o valor 7, e unindo os dois dígitos temos 47. Agora passe para a terceira faixa (laranja), que é o multiplicador 1 kΩ. Com base nesses valores, podemos calcular o seu valor 47 x 1 kΩ, ou seja, 47 kΩ. A quarta faixa, dourada, indica a tolerância de valor do componente, isto é, 5%.

Cor	Dígito 1ª faixa	Dígito 2ª faixa	Multiplicador 3ª faixa	Tolerância 4ª faixa
Preto	0	0	1 Ω	–
Marrom	1	1	10 Ω	–
Vermelho	2	2	100 Ω	–
Laranja	3	3	1 kΩ	–
Amarelo	4	4	10 kΩ	–
Verde	5	5	100 kΩ	–
Azul	6	6	1 MΩ	–
Violeta	7	7	10 MΩ	–
Cinza	8	8	–	–
Branco	9	9	–	–
Ouro	–	–	–	± 5%
Prata	–	–	–	± 10%

Os **capacitores** são componentes que permitem armazenar energia para uma utilização rápida. Por exemplo, se compararmos um capacitor com uma pilha, temos que o capacitor pode descarregar toda a sua carga em uma pequena fração de segundo, enquanto a pilha demoraria vários minutos para isso. Uma aplicação típica para os capacitores é no circuito eletrônico que ativa o flash de uma câmera fotográfica, a pilha (ou bateria) carrega o capacitor por vários segundos, e então o capacitor descarrega toda a carga armazenada para que a lâmpada do flash seja acionada imediatamente. Existem diversos tipos de capacitores, alguns polarizados e outros não. A unidade de medida de um capacitor é o farad (F).

Existem muitos tipos de capacitores. A Figura 2.7 ilustra os modelos mais comumente encontrados nos circuitos eletrônicos: à esquerda temos o capacitor cerâmico, que não é polarizado e é capaz de armazenar pequenas quantidades de energia; à direita podemos observar o capacitor eletrolítico, que é polarizado e tem a capacidade de armazenar quantidades maiores de energia.



Figura 2.7 – Capacitores.

Um **diodo** é um componente semicondutor que permite que a corrente flua em apenas um sentido. Os semicondutores são elementos que apresentam características elétricas entre condutores e isolantes, ou seja, podem alternar o seu comportamento de condutor para isolante e vice-versa. Podemos entender como materiais isolantes aqueles que não permitem a passagem de corrente elétrica, como, por exemplo, a madeira ou o plástico. Por outro lado, materiais condutores são aqueles que possibilitam a passagem de corrente elétrica quando é aplicada uma diferença de potencial. Podemos citar como exemplos o cobre, a prata e o ouro. O cristal de silício é o semicondutor mais utilizado na fabricação de diversos componentes eletrônicos além dos diodos. Mais raro e bem menos utilizado temos também o germânio. Conforme podemos observar na Figura 2.8 o diodo possui dois terminais: o terminal identificado com uma faixa é chamado cátodo e deve estar sempre conectado ao polo negativo da alimentação; o outro é o ânodo e deve estar conectado ao polo positivo da alimentação.



Figura 2.8 – Diodo.

O **diodo emissor de luz**, ou simplesmente **LED**, é uma variação do diodo e apresenta como principal característica a emissão de luz quando uma corrente flui através dele. É um componente polarizado. Dessa forma, conforme podemos observar na Figura 2.9, o cátodo, que é o lado chanfrado e apresenta um terminal (perna) mais curto, sempre deve estar conectado ao polo negativo (ou terra) do circuito. Se conectado invertido, o LED não funcionará e também poderá ser danificado.



Figura 2.9 – Diodo emissor de luz (LED).

Os **transistores** são componentes semicondutores e foram os principais responsáveis pela revolução da eletrônica e da informática na década de 1960, pois permitiram substituir as válvulas nos equipamentos. Um transistor é praticamente cem vezes menor que uma válvula, não necessita de tempo para aquecimento, consome menos energia, além de ser muito mais rápido e confiável. Apresenta inúmeras aplicações, e as principais são a atuação como uma "chave" eletrônica e amplificador de sinais elétricos. Na Figura 2.10 podemos observar um transistor de uso geral.



Figura 2.10 – Transistor de uso geral.

Um transistor possui três terminais, denominados base, coletor e emissor e, para identificá-los, devemos ter o modelo utilizado e consultar um manual com as respectivas especificações.

Os **circuitos integrados** consistem em transistores e vários outros componentes eletrônicos miniaturizados e montados num único chip. A integração em larga escala permite colocar cerca de 1 milhão de transistores por mm^2, proporcionando um alto nível de miniaturização dos circuitos eletrônicos, além de uma grande redução de custos. Conforme ilustrado pela Figura 2.11, um circuito integrado possui um chanfro em formato de meia-lua que indica o posição do pino 1. Devemos identificar o modelo do circuito integrado e, em seguida, consultar a função de cada pino no respectivo manual.



Figura 2.11 – Circuito integrado.

Uma protoboard permite a montagem provisória de circuitos eletrônicos, possibilitando a reutilização dos componentes. Consiste em uma matriz de contatos interconectados através dos quais os componentes são interligados. Conforme ilustrado na Figura 2.12, um protoboard tipicamente está organizado em "ilhas" de contatos. As linhas mais externas são utilizadas para alimentação, e ele apresentará uma linha para o

polo positivo, normalmente indicada por uma contínua vermelha, e outra para o polo negativo, normalmente indicada em azul ou preto. Dessa forma, sempre que desejarmos alimentar um determinado componente do circuito com uma tensão positiva, basta conectá-lo a qualquer contato que esteja indicado com a linha vermelha. Por outro lado, quando é necessário conectar um componente ao polo negativo da alimentação, basta utilizar qualquer um dos contatos indicados com a linha azul ou preta.

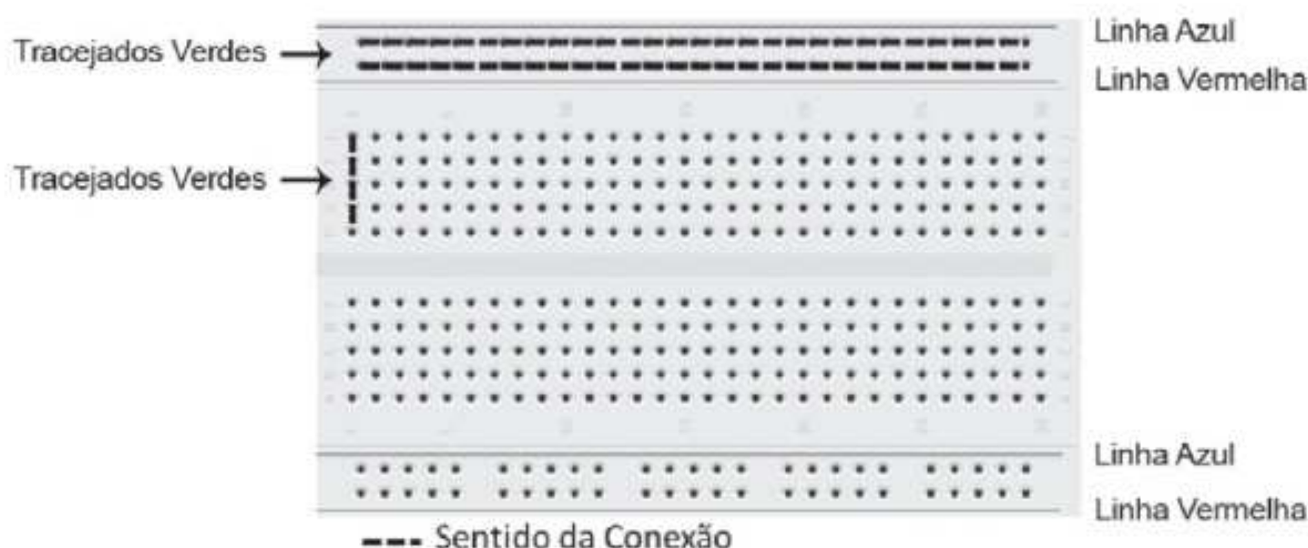


Figura 2.12 – Protoboard.

As duas ilhas centrais são destinadas a receber os componentes que serão utilizados no circuito que está sendo montado. As colunas em uma mesma linha estão ligadas entre si. Dessa forma, dois ou mais componentes que precisam estar conectados devem ter os seus terminais ocupando uma mesma coluna. As linhas, por sua vez, não apresentam conexão entre si. Um mesmo componente nunca pode ter os seus terminais ocupando uma mesma coluna de uma mesma ilha, pois, dessa forma, seus terminais sempre estarão com um mesmo potencial, a corrente elétrica não fluirá e o componente não funcionará.

As conexões entre o Arduino, os componentes e a protoboard também podem ser realizadas por meio de **jumper cable** (fio), ilustrado na Figura 2.13, que nada mais é do que um pequeno pedaço de fio metálico que permite a conexão.



Figura 2.13 – Jumper cable.

Exercícios

1. Pesquise e dê exemplos da utilização dos componentes relacionados a seguir. Por exemplo, cite quais aparelhos eletrônicos os utilizam e quais são as suas principais finalidades.

a) LEDs.

b) Diodos.

c) Capacitores.

d) Resistores.

e) Transistores.

2. Descreva a sequência de cores para os seguintes resistores:

a) 220 Ω.

b) 330 Ω.

c) 1 kΩ.

d) 10 kΩ.

3. Pela Lei de Ohm, qual é a tensão necessária para obtermos uma corrente de 100 mA (0,1 A), considerando uma resistência de 10 kΩ?

4. Aplicando a Lei de Ohm, qual é a resistência necessária para obtermos uma corrente de 1 A a partir de uma fonte de alimentação que forneça 50 volts?

5. Uma resistência de 200 Ω é submetida a uma tensão de 100 volts. Aplicando a Lei de Ohm, determine a intensidade de corrente que passará pela resistência.

3

Montagem do Primeiro Projeto

Neste capítulo teremos nosso primeiro contato com a plataforma, tanto em relação aos componentes de hardware quanto ao software utilizado para a construção dos programas para o Arduino. Visando facilitar o processo de aprendizagem, todos os projetos deste livro seguem um determinado roteiro de montagem, o qual apresentará os seguintes passos:



Material necessário

Este tópico apresentará os componentes necessários para a montagem do projeto.



Montagem do circuito

A montagem do circuito apresentará o diagrama de montagem, tomando como base o uso da protoboard, além dos passos detalhados para a conexão dos componentes utilizados.



Programa

No tópico Programa será apresentado o código-fonte do programa, além da explicação sobre o seu funcionamento. Como se trata de uma plataforma programável, note que um mesmo projeto (hardware) pode apresentar vários programas diferentes, os quais produzirão um funcionamento variado de um mesmo circuito eletrônico.

Nos Capítulos 11 e 12 é abordada a integração do Arduino com outras linguagens de programação, no caso, Processing e Java. Processing será usada para montar aplicações em um computador, enquanto Java será utilizada para o desenvolvimento de aplicativos para Android. Dessa forma, nos roteiros do Capítulo 11, o símbolo mostrado a seguir identificará um programa escrito em Processing.



Programa

No Capítulo 12, o símbolo a seguir, por sua vez, identificará o código-fonte utilizado no desenvolvimento de um aplicativo Android.



Programa



PROJETO Nº 1
Controle de um LED

O objetivo deste primeiro projeto é utilizar uma porta digital do Arduino para controlar o funcionamento de um diodo emissor de luz (LED – *Light Emitting Diode*). Um nível 1 (HIGH) colocado no pino acenderá o LED, enquanto um nível 0 (LOW) vai apagar o LED.

3.1 Identificação dos componentes

O primeiro passo para a montagem do projeto consiste em realizar a correta identificação dos componentes que serão necessários (Figura 3.1).



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.

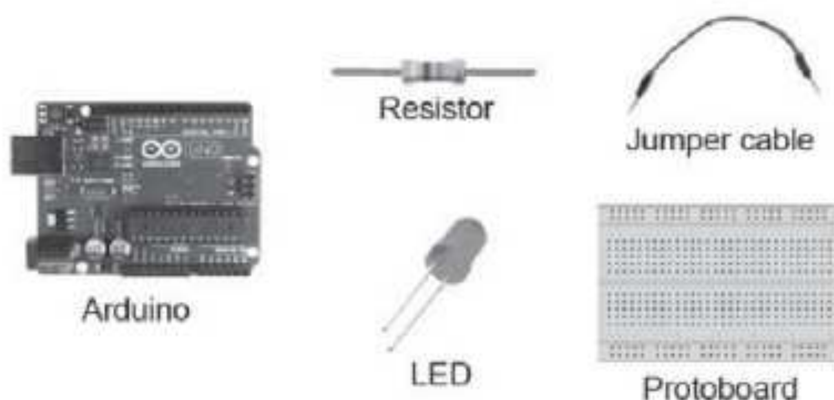


Figura 3.1 – Componentes utilizados no projeto.

3.2 Montagem do circuito

Como vimos anteriormente, os projetos montados neste livro utilizarão um protoboard de modo a permitir a reutilização posterior dos componentes aplicados, além de facilitar o processo de alteração do circuito e a correção dos erros de montagem que, porventura, possam ocorrer.



Montagem do circuito

Na Figura 3.2 temos a maneira como esse primeiro projeto deverá ser montado. Inicialmente, identifique e separe os componentes que serão utilizados. No caso do LED, tenha uma atenção especial para identificar os pinos corretos, ou seja, ânodo e cátodo, pois caso seja montado de maneira invertida, o LED não acenderá. Também identifique o valor do resistor com base nas faixas coloridas que estão presentes no corpo do mesmo.

Observe também que o esquema apresentado na Figura 3.2, assim como os demais desenhos esquemáticos apresentados no livro, foi feito por meio de uma aplicação denominada Fritzing. Essa útil ferramenta pode ser baixada gratuitamente em seu site oficial (http://www.fritzing.org/). No Fritzing é possível criar um esboço dos diagramas que serão utilizados nos projetos, incluindo a placa Arduino, a protoboard e os diversos componentes eletrônicos utilizados. No Capítulo 13 apresentamos um breve tutorial sobre como utilizar o Fritzing.

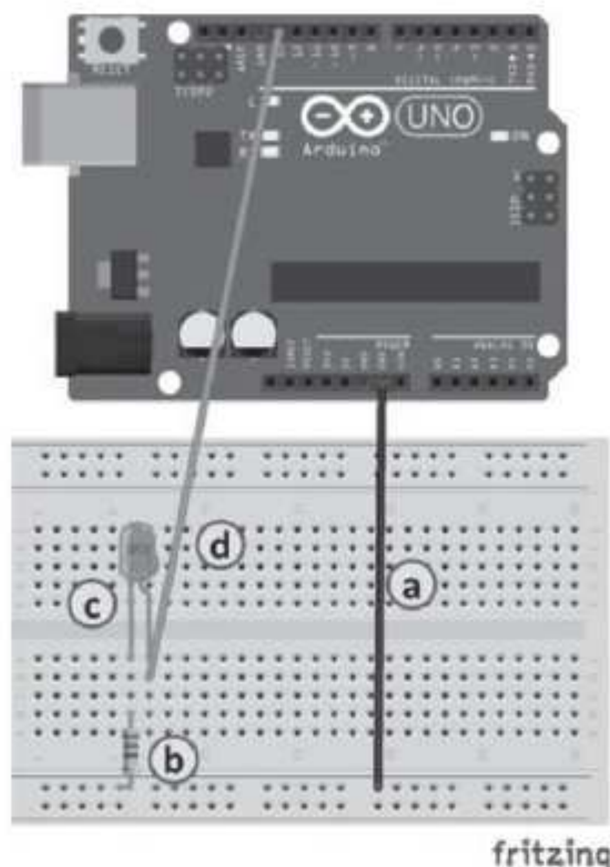


Figura 3.2 – Controle de um LED.

Dessa forma, conforme ilustra a Figura 3.2:

a) Usando o jumper cable, conecte um dos pinos de GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.

b) Coloque um terminal (extremidade) do resistor de 220 ohms (ou 330 ohms) em qualquer posição da linha de alimentação negativa (preta ou azul). O outro terminal do resistor deve ser conectado a qualquer outra linha da protoboard que não seja reservada para a alimentação positiva do circuito (linha vermelha) ou negativa (preta ou azul).

c) Coloque o LED com o cátodo (terminal mais curto e lado chanfrado) conectado na mesma coluna da ilha da protoboard na qual está ligado o terminal do resistor.

d) Conecte um jumper cable na mesma coluna na qual estão ligados o ânodo do LED e o pino 13 do Arduino.

3.3 Sketch

Um programa escrito para o Arduino é denominado sketch, e, conforme abordado anteriormente, o programa é desenvolvido em um ambiente de desenvolvimento próprio. Uma vez concluída a compilação do sketch, ele deve ser transferido para o Arduino, no qual entrará em um loop de execução infinito.



Após a montagem dos componentes na protoboard, devemos agora realizar o desenvolvimento do programa que interagirá com o circuito. Para fazer isso, inicie o ambiente de desenvolvimento do Arduino e digite o seguinte sketch:

```
// Controlar um LED através da saída digital

int LED = 13;                // Pino no qual o LED está conectado

void setup() {
  pinMode(LED, OUTPUT);      // Definir o pino como saída
}

void loop() {
  digitalWrite(LED, HIGH);   // Acender o LED (Nível 1 no pino)
  delay(2000);                        // Aguardar por 2 segundos
  digitalWrite(LED, LOW);    // Nível 0 no pino, apagar o LED
  delay(2000);
}
```

Conforme ilustrado na Figura 3.3, após salvar o sketch digitado, faça a compilação e, em seguida, conecte o Arduino à porta USB do computador. Por último, pressione o botão Transferir para realizar a transferência do sketch para a memória flash do Arduino.

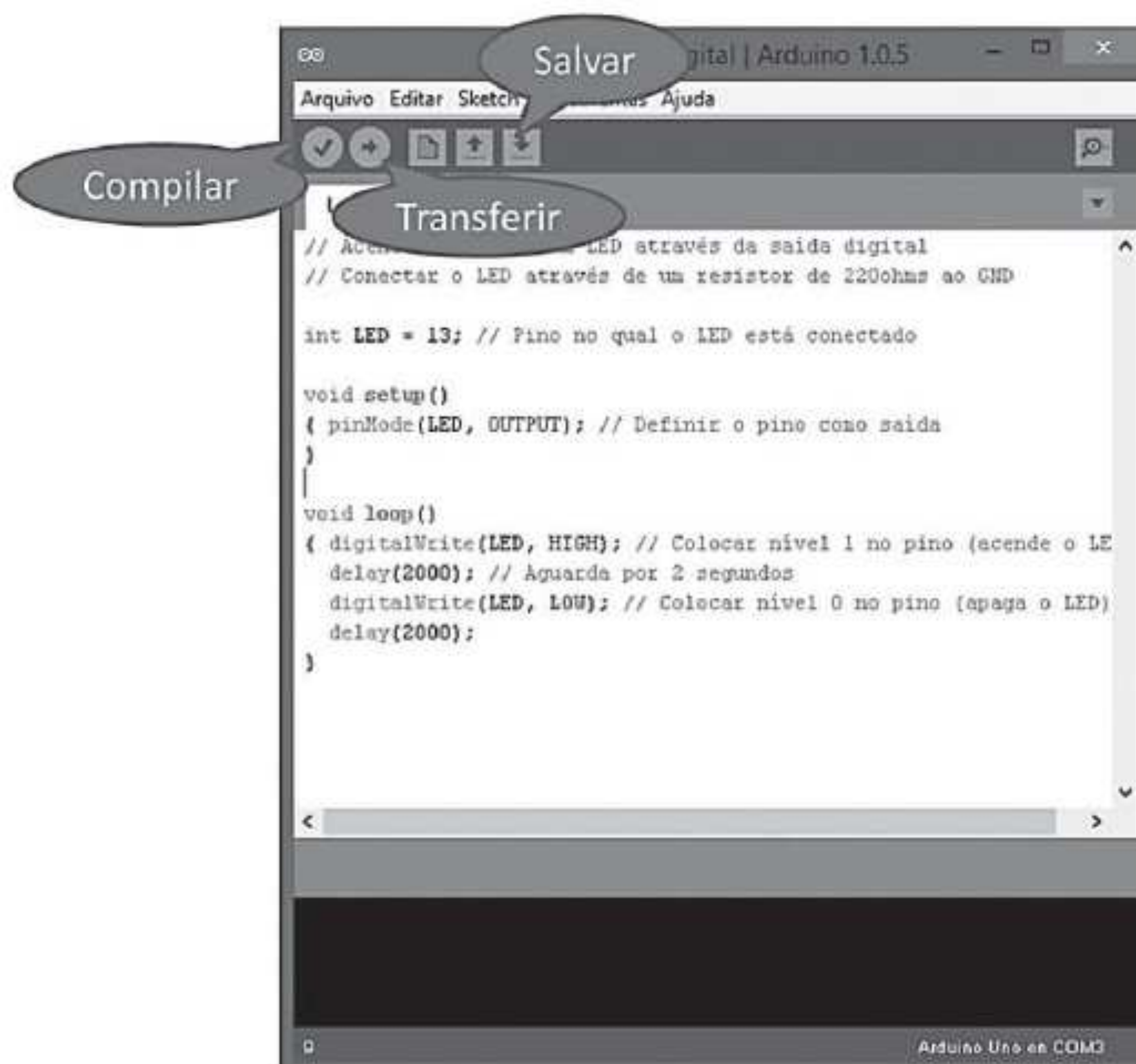


Figura 3.3 – Botões para compilar, transferir e salvar.

Também é importante salientar que um sketch ficará armazenado na memória do Arduino, mesmo quando este for desligado. O sketch é substituído somente quando ocorre uma nova transferência, podendo ser uma nova versão do mesmo sketch ou um novo programa. Apenas um sketch fica armazenado por vez na memória e permanece em execução contínua enquanto o Arduino está ligado.

Agora vamos passar para a análise do sketch que foi desenvolvido e entender a estrutura básica de um programa para o Arduino. Note que primeiro criamos uma variável chamada LED, que armazenará o número do pino ao qual o LED está conectado.

```
int LED = 13;
```

Conforme podemos observar no trecho de programa a seguir, o passo seguinte é informar que o pino 13 será uma saída. Fazemos isso por meio da função pin mode. Nela vamos definir qual o pino a ser configurado, que nesse exemplo é o 13, ou seja, é o pino no qual o LED está conectado. Também devemos informar qual será a sua função, que nesse exemplo será uma saída (OUTPUT).

```
void setup(){
  pinMode(LED, OUTPUT);
}
```

Nesse mesmo trecho de programa é possível observar a utilização da função setup, que tem como objetivo realizar as configurações iniciais do programa e é executada apenas uma vez quando o Arduino é ligado ou reiniciado.

A seguir utilizamos uma função para "escrever", ou seja, enviar um sinal elétrico para o pino ao qual o LED está conectado, com a função digitalWrite. Acender o LED significa enviar uma tensão de 5 volts para o pino 13, alimentando assim o circuito que controla o acendimento do LED. Isso ocorre quando utilizamos a constante **HIGH**, que indica a passagem de 5 volts para o pino. Quando é necessário apagar o LED, passamos a constante LOW, que colocará zero volts no pino, apagando-o. Mais detalhes dessas constantes serão apresentados no Capítulo 4. Como podemos observar no trecho de programa a seguir, a função digitalWrite recebe dois parâmetros, que são o pino no qual será colocado o sinal elétrico e qual valor a ser escrito, ou seja, HIGH ou LOW.

```
void loop() {
  digitalWrite(LED, HIGH);
  delay(2000);
  digitalWrite(LED, LOW);
  delay(2000);
}
```

Outra função utilizada neste exemplo é a delay, que apresenta como objetivo criar um atraso na execução do programa, valor este expresso em milissegundos (ms). Nesse exemplo, após acender o LED, colocamos um atraso de 2 segundos, ou seja, 2.000 milissegundos, e depois o apagamos, permanecendo nesse estado por mais 2 segundos.

Esse evento de acender, esperar, apagar e esperar será repetido ciclicamente, ou seja, repetidas vezes, enquanto o programa estiver executando. Isso ocorre devido à função loop. Todo sistema de automação nada mais é do que um conjunto de sensores e atuadores que periodicamente possuem seus estados verificados e, quando necessário, alterados. As funções setup e loop são essenciais e obrigatórias para qualquer programa desenvolvido para o Arduino.

Exercícios

1. Altere o sketch do Projeto nº 1 para manter o LED aceso por 4 segundos e, em seguida, apagado por 1 segundo.
2. Adicione ao Projeto nº 1 um outro resistor de 220 ohms e um LED conectado ao pino 12 do Arduino. Altere o sketch de modo a alternar o acendimento dos LEDs a cada 1 segundo.
3. Considerando que o projeto agora está com dois LEDs conectados, crie um sketch que faça um contador binário, sendo que o intervalo entre cada estado deve ser de 1 segundo. Ou seja, inicialmente, os dois LEDs devem estar apagados, após 1 segundo um LED deve ser acesso, em seguida este é apagado e o outro deve ser aceso, e, por último, os dois LEDs devem estar acesos.

4 Entradas e Saídas

O Arduino oferece um conjunto de pinos que permitem a entrada e saída de dados. Os dados de entrada são enviados por componentes (ou módulos) chamados sensores, como um botão liga-desliga ou um sensor de temperatura. A saída de dados permite definir o comportamento dos dispositivos atuadores, como, por exemplo, um LED, um buzzer ou um alto-falante.

Uma entrada ou saída digital trabalha com apenas dois valores: 0 (LOW) ou 1 (HIGH). Por outro lado, as entradas analógicas trabalham com uma faixa de valores intermediários entre o valor mínimo e o máximo. No Arduino, uma entrada analógica trabalha com os valores inteiros entre 0 e 1.023. Para obter um sinal de saída entre 0 e 255, o Arduino utiliza o conceito de Modulação por Largura de Pulso (*Pulse Width Modulation* – PWM) para obter um sinal de saída inteiro que varia entre 0 e 255.

4.1 Saída digital

Como visto, o Arduino apresenta 14 portas digitais que podem ser utilizadas tanto para entrada como para saída. Nas portas digitais, são utilizados apenas dois valores, ou seja, 0 (LOW) ou 1 (HIGH).



PROJETO Nº 2
Semáforo

O circuito que montamos anteriormente já é um exemplo de utilização de uma saída digital no Arduino. Vamos agora criar um novo circuito que simulará o funcionamento de um sinal de trânsito. Para isso, utilizaremos três LEDs, cada um controlado por um pino do Arduino.



Material necessário

- 1 Arduino;
- 3 resistores de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED vermelho;
- 1 LED verde;
- 1 LED amarelo;
- 1 protoboard;
- jumper cable.



Montagem do circuito

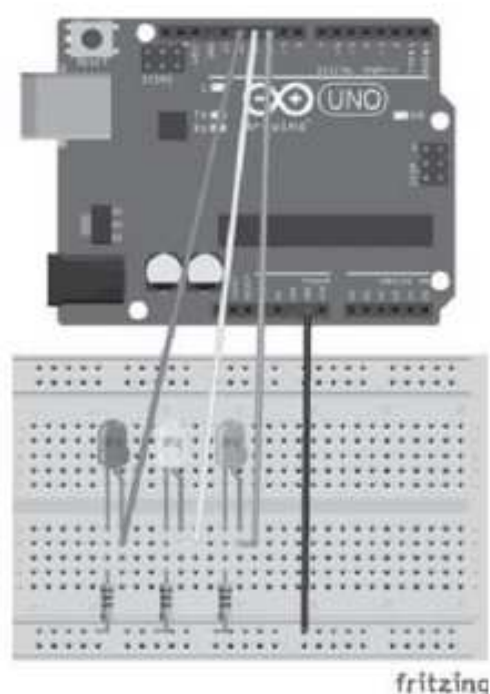


Figura 4.1 – Projeto do semáforo (para versão colorida, consulte o site da Editora Érica).

Siga estes passos para montar o circuito, conforme ilustra a Figura 4.1:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Coloque um dos resistores de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra linha da protoboard.
c) Coloque o LED vermelho com o cátodo (lado chanfrado) conectado ao resistor.
d) Conecte o ânodo do LED ao pino 12 do Arduino.
e) Coloque o outro resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer linha disponível da protoboard.
f) Coloque o LED amarelo com o cátodo (lado chanfrado) conectado ao resistor.

g) Conecte o ânodo do LED ao pino 11 do Arduino.

h) Coloque o terceiro resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer linha não utilizada da protoboard.

i) Coloque o LED verde com o cátodo (lado chanfrado) conectado ao resistor.

j) Conecte o ânodo do LED ao pino 10 do Arduino.



Programa

Após montar o circuito do projeto, vá até o ambiente de desenvolvimento do Arduino e digite o programa a seguir:

```
// Simulação de um Semáforo
// Conectar os LEDs através de um resistor de 220 ohms ao GND

int VERD = 10;
int AMAR = 11;
int VERM = 12;

void setup() {
  pinMode(VERD, OUTPUT);
  pinMode(AMAR, OUTPUT);
  pinMode(VERM, OUTPUT);
}

void loop() {
  digitalWrite(VERM, HIGH);
  digitalWrite(AMAR, LOW);
  digitalWrite(VERD, LOW);
  delay(2000);

  digitalWrite(VERM, LOW);
  digitalWrite(AMAR, LOW);
  digitalWrite(VERD, HIGH);
  delay(3000);

  digitalWrite(VERM, LOW);
  digitalWrite(AMAR, HIGH);
  digitalWrite(VERD, LOW);
  delay(1000);
}
```

O funcionamento desse projeto é bem simples: os três pinos nos quais os LEDs, nossos **atuadores**, estão conectados, são definidos como saída (OUTPUT) através do uso da função pinMode, ou seja:

```
void setup() {
  pinMode(VERD, OUTPUT);
  pinMode(AMAR, OUTPUT);
  pinMode(VERM, OUTPUT);
}
```

No laço de execução do programa (loop), o LED vermelho ficará aceso por 2 segundos, enquanto os demais ficam apagados:

```
digitalWrite(VERM, HIGH);
digitalWrite(AMAR, LOW);
digitalWrite(VERD, LOW);
delay(2000);
```

Após esse intervalo de tempo, o LED verde será aceso enquanto os outros dois ficam apagados por 3 segundos:

```
digitalWrite(VERM, LOW);
digitalWrite(AMAR, LOW);
digitalWrite(VERD, HIGH);
delay(3000);
```

Completando o ciclo, o LED amarelo ficará aceso por 1 segundo enquanto os demais estarão apagados:

```
digitalWrite(VERM, LOW);
digitalWrite(AMAR, HIGH);
digitalWrite(VERD, LOW);
delay(1000);
```

Após isso, o laço do programa voltará ao seu início, ou seja, o LED vermelho será acesso e os demais ficarão apagados e assim sucessivamente.

4.2 Entrada digital

A entrada digital é utilizada para enviarmos ao Arduino um valor digital 0 (LOW) ou 1 (HIGH). Qualquer um dos 13 pinos digitais pode ser programado como entrada digital.



PROJETO Nº 3
Controle de um LED a partir de um botão

Neste projeto utilizaremos um botão de pressão (push button) para controlar o acender e apagar de um LED. O botão será o nosso dispositivo de entrada, ou seja, o sensor, enquanto o LED será o de saída, isto é, o atuador.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10 kohms (marrom, preto, laranja);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

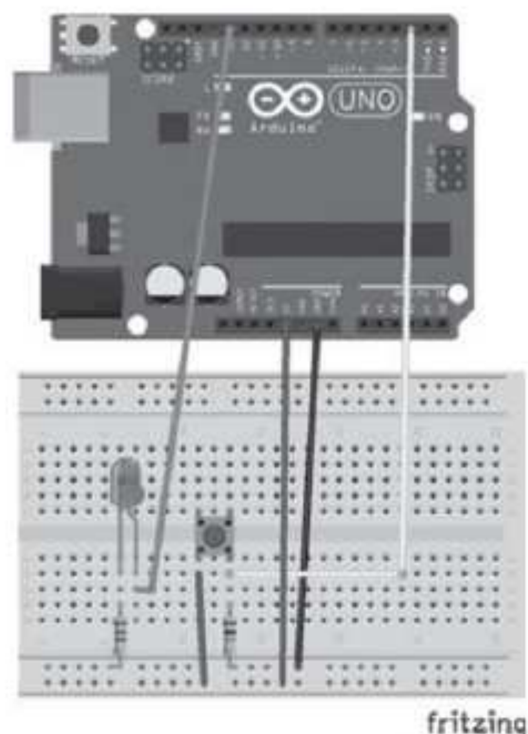


Figura 4.2 – Uso do interruptor de pressão.

Conforme ilustra a Figura 4.2, devemos realizar os seguintes passos para montar o projeto:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.

b) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
c) Conecte um terminal do botão ao pino digital 2 do Arduino.
d) Conecte o resistor de 10 kohms entre o mesmo pino do botão usado no item anterior e a linha de alimentação negativa (preta) da protoboard.
e) Conecte outro terminal do botão à linha de alimentação positiva (vermelha) da protoboard.
f) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
g) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
h) Conecte o ânodo do LED ao pino 13 do Arduino.



Programa

Digite o seguinte sketch no ambiente de desenvolvimento do Arduino:

```
int LED = 13;             // Pino para o LED
int BOTAO = 2;            // Pino de entrada (para o botão)
int valorEntrada = LOW;   // Obtém o estado do botão

void setup() {
  pinMode(LED, OUTPUT);   // Definir o LED como saída
  pinMode(BOTAO, INPUT); // Definir o botão como entrada
}

void loop(){
  valorEntrada = digitalRead(BOTAO); // Obter o valor de entrada
  if (valorEntrada == HIGH) {
    digitalWrite(LED, LOW); // Desligar o LED
  }
  else {
    digitalWrite(LED, HIGH); // Ligar o LED
  }
  delay(100);
}
```

Compile o programa e o envie ao Arduino, pressione momentaneamente o botão e observe a mudança de estado do LED. O botão está conectado entre as linhas positiva (HIGH) e negativa (LOW). Dessa forma, quando não pressionado, ele mantém o nível

LOW na entrada digital do Arduino, mantendo, conforme o programa, o LED aceso. Ao ser pressionado, o nível é alterado para HIGH e o LED ficará apagado por alguns instantes.

Você deve ter observado que, nesta experiência, ainda não é possível alterar o estado do botão de forma indefinida, ou seja, até que ocorra um novo pressionamento dele. Para obter isso, devemos armazenar o estado do botão e trocá-lo a cada pressionamento. No sketch a seguir, que utilizará o mesmo circuito, vamos incorporar a função de liga-desliga ao botão.



Programa

Mantenha a mesma montagem utilizada no projeto anterior, porém crie um novo sketch a partir do seguinte código-fonte:

```
int LED = 13;          // Pino para o LED
int BOTAO = 2;         // Pino de entrada (para o botão)
int valorEntrada = LOW;  // Obter o estado do botão
int valorAnterior = LOW; // Manter o estado anterior do botão
int estado = LOW;

void setup() {
  pinMode(LED, OUTPUT);  // Utilizar o LED como saída
  pinMode(BOTAO, INPUT); // Utilizar o botão como entrada
}

void loop(){
  // Obter o valor de entrada
  valorEntrada = digitalRead(BOTAO);

  // Verificar se houve mudança de estado do botão
  if ((valorEntrada == HIGH) && (valorAnterior == LOW)) {
    estado = 1 - estado;
    delay (10);
  }

  if (estado == LOW) {
    digitalWrite(LED, LOW);  // Desligar o LED
  }
  else {
    digitalWrite(LED, HIGH);  // Ligar o LED
}

  valorAnterior = valorEntrada;
}
```

Nesse sketch você pode observar que foi declarada uma variável para manter o estado do botão e, a cada pressionamento do botão, esse estado é invertido, ou seja, se a variável estado está armazenando o valor LOW, ao pressionarmos o botão o valor será alterado para HIGH e vice-versa, proporcionando, dessa forma, o efeito de liga e desliga sobre o LED.

4.3 Saída PWM

Nas saídas PWM, que consistem nos pinos digitais 3, 5, 6, 9, 10 e 11 e que estão identificados com um sinal de til (~) ou pela sigla PWM, podemos ter um valor entre 0 e 255.

A PWM consiste em uma técnica que permite, a partir de um sinal analógico, produzir um conjunto de sinais digitais enviados durante um determinado período. Se plotarmos em um gráfico um sinal digital ao longo do tempo, vamos obter uma forma de onda quadrada que alterna seu estado em nível lógico 1 (HIGH) e 0 (LOW).

A razão entre o período de pico e o total da onda é chamada de razão de ciclo (duty cycle). Dessa forma, podemos entender que, se durante um período temos apenas o nível lógico 0 (LOW), a saída PWM produzirá uma razão de ciclo de 0%, resultando no valor zero, conforme podemos observar na Figura 4.3:



Figura 4.3 – Razão de ciclo de 0%.

Por outro lado, se durante todo o período o sinal permanecer apenas um nível 1 (HIGH), teremos 100% da razão de ciclo (Figura 4.4), resultando no valor 255 se considerarmos, nesse exemplo, uma saída digital do Arduino.

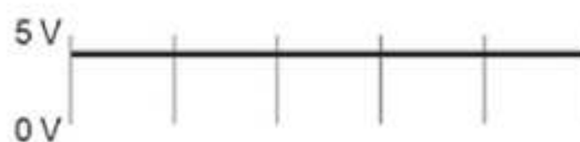


Figura 4.4 – Razão de ciclo de 100%.

Do mesmo modo, podemos obter valores intermediários entre esses dois extremos. Por exemplo, se a razão de ciclo for 25% do tempo em nível 1 (HIGH) e 75% do tempo em nível 0 (LOW), teremos uma razão de ciclo de 25% (Figura 4.5), a qual resultará no valor 64, ou seja, um quarto (25%) de 256.



Figura 4.5 – Razão de ciclo de 25%.



PROJETO Nº 4
LED com efeito pulsante

O objetivo deste projeto é utilizar uma saída PWM para controlar a intensidade luminosa de um LED.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

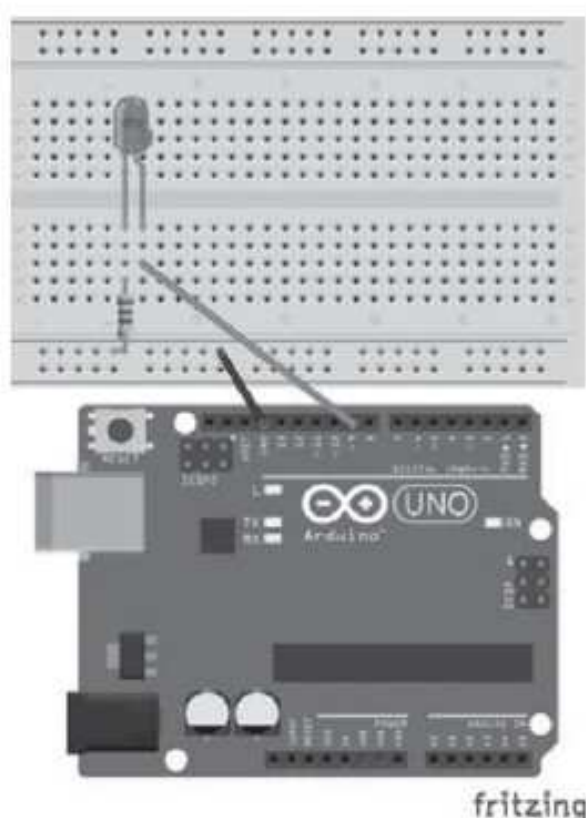


Figura 4.6 – LED pulsante.

Conforme ilustra a Figura 4.6:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
c) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
d) Conecte o ânodo do LED ao pino 9 do Arduino.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o sketch (programa) a seguir:

```
// LED Pulsante
// Controlar a luminosidade de um LED através da saída PWM
// Conectar o LED através de um resistor de 220 ohms ao GND

int LED = 9; // Pino ao qual o LED está conectado
int i = 0;

void setup() {
  pinMode(LED, OUTPUT); // Definir o pino como saída
}

void loop() {
  for (i = 0; i < 255; i++) {
    analogWrite(LED, i); // Define o nível de luminosidade do LED
  delay(10); // Aguarda por 10 milissegundos
  }
  for (i = 255; i >= 0; i--) {
    analogWrite(LED, i);
    delay(10);
  }
}
```

Nesse sketch, o pino 9 será utilizado como uma saída PWM. Todos os pinos do Arduino que possuem o símbolo ~ (til) ou a sigla PWM na frente do número indicam que aquele determinado pino pode ser utilizado tanto em modo digital quanto analógico. No laço do programa vamos fazer um primeiro utilizando o comando for, que gradativamente

aumentará a intensidade luminosa do LED até atingir o seu valor máximo, ou seja, 255. Observe o uso da função analogWrite para podermos enviar um valor analógico.

Em seguida, o segundo laço for fará o contrário: partindo do valor máximo (255), gradativamente reduzirá esse valor e, por consequência, a intensidade luminosa do LED até o valor mínimo (0). A repetição desses dois laços pelo programa gerará um efeito de pulsação do LED.

O exemplo a seguir produzirá um efeito similar, porém, neste caso, vamos utilizar o uso da função de seno (sin) para determinar o valor analógico que será aplicado ao LED. Nesse caso observe que foi necessário apenas um laço de repetição, pois a variação do valor do seno entre os ângulos 0 até 180 graus já produz o efeito que desejamos.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o sketch a seguir:

```
// LED Pulsante
// Controlar a luminosidade de um LED através da saída PWM
// Conectar o LED através de um resistor de 220 ohms ao GND

int LED = 9;
float seno;
int intensidade;

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  for (int i = 0; i < 180; i++) {
    seno = sin(i * (3.1416 / 180));
    intensidade = int(seno * 255);
    analogWrite (LED, intensidade);
    delay (25);
  }
}
```

4.4 Entrada analógica

Nas seis portas analógicas de entrada presentes no Arduino, podemos receber um valor inteiro entre 0 e 1023 proveniente de um sensor.



PROJETO Nº 5
Controle de um LED por meio de potenciômetro

O objetivo deste projeto é controlar a frequência de acendimento de um LED. Um potenciômetro nada mais é que um resistor variável no formato de um botão giratório, que fornece valores intermediários de resistência entre zero ohm e a máxima especificada no corpo do componente. Utilizaremos o potenciômetro para atuar como um sensor que produzirá uma sequência de valores em uma das entradas analógicas do Arduino.



Material necessário

- 1 Arduino;
- 1 potenciômetro;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

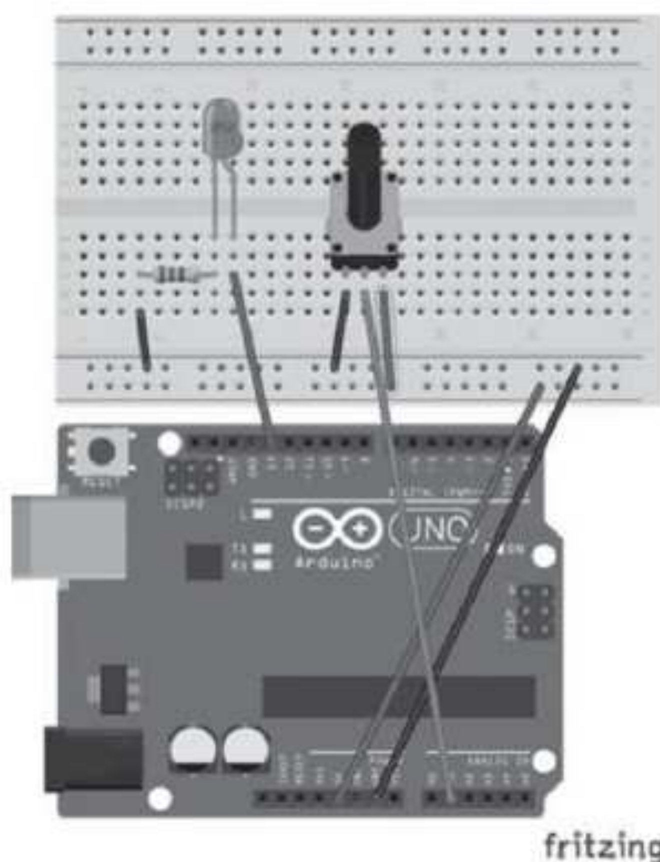


Figura 4.7 – Entrada analógica.

Conforme ilustra a Figura 4.7, devemos realizar os seguintes passos para montar o circuito:

a) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
c) Conecte um LED utilizando um resistor de 220 ohms (ou 330 ohms).
d) Conecte o LED no pino digital 13.
e) Conecte o potenciômetro na protoboard conforme a Figura 4.7 (botão de girar virado para você).
f) Conecte o pino da esquerda do potenciômetro na linha de alimentação GND.
g) Conecte o pino da direita do potenciômetro na linha de alimentação positiva.
h) Conecte o pino do centro do potenciômetro no pino analógico A1 do Arduino.



Programa

No ambiente de desenvolvimento do Arduino, digite o seguinte programa:

```
//Liga e desliga um LED na frequência determinada
// pelo potenciômetro.

int POT = A1;   // Pino de entrada conectado ao potenciômetro
int LED = 13;   // Pino de saída conectado ao LED
int valor = 0;  // Armazenar o valor do potenciômetro

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(POT, INPUT);
}

void loop() {
  valor = analogRead(POT);
  digitalWrite(LED, HIGH);
  delay(valor);
  digitalWrite(LED, LOW);
  delay(valor);
}
```

Quando giramos o potenciômetro, alteramos a resistência em cada lado do contato elétrico que vai conectado ao terminal central do botão. Essa mudança implica uma alteração

no valor analógico de entrada. Quando o cursor for levado até o final da escala, teremos 0 V, assim obtendo o valor zero na entrada analógica. Quando giramos o cursor até o outro extremo da escala, teremos 5 V, gerando o valor 1.023 na entrada analógica.



PROJETO Nº 6
Controle da luminosidade de um LED

O objetivo deste projeto é controlar a intensidade luminosa de um LED a partir dos valores analógicos obtidos por meio de um potenciômetro.



Material necessário

- 1 Arduino;
- 1 potenciômetro;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

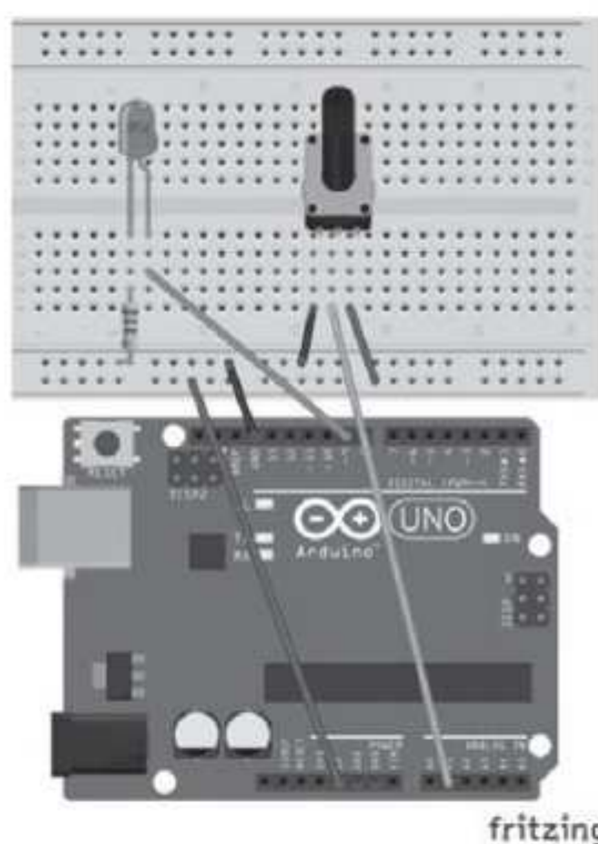


Figura 4.8 – Entradas e saídas digitais.

Conforme ilustra a Figura 4.8:

a) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
c) Conecte um LED utilizando um resistor de 220 ohms (ou 330 ohms).
d) Conecte o LED no pino analógico 9.
e) Conecte o potenciômetro na protoboard conforme a Figura 4.8 (botão de girar virado para você).
f) Conecte o pino da esquerda do potenciômetro na linha de alimentação GND.
g) Conecte o pino da direita do potenciômetro na linha de alimentação positiva.
h) Conecte o pino do centro do potenciômetro no pino analógico A1 do Arduino.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o seguinte código-fonte:

```
//Controla a intensidade luminosa de um LED através do
// uso de um potenciômetro conectado a uma entrada analógica do
// Arduino

// Pino analógico do Arduino ao qual será ligado o potenciômetro
int POT = A1;

// Pino PWM do Arduino ao qual será conectado o LED
int LED = 9;

int valor = 0;

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  valor = analogRead(POT);
  if(valor > 0){
    //Acende o LED com intensidade proporcional ao valor obtido
    analogWrite(LED, (valor/4));
  }
}
```

Nesse sketch, a partir do valor do potenciômetro enviado para a entrada analógica, ou seja, obtido pela função analogRead, definimos a intensidade luminosa do LED através da função analogWrite. Observe que, como a entrada recebe um valor entre 0 a 1.023 e a saída trabalha com valores entre 0 e 255, o valor obtido é dividido por 4 para termos a devida proporção.



PROJETO Nº 7
Controle da luminosidade de um LED por meio de botão

O objetivo deste projeto é controlar a luminosidade de um LED por meio de uma saída PWM. O LED começa com seu estado apagado. Com um pressionar no botão, altera-se o estado do LED para aceso. Caso o botão permaneça pressionado por mais de 5 segundos, ocorrerá a variação de luminosidade do LED.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10 kohms (marrom, preto, laranja);
- 1 LED;
- 1 pushbutton;
- 1 protoboard;
- jumper cable.



Montagem do circuito

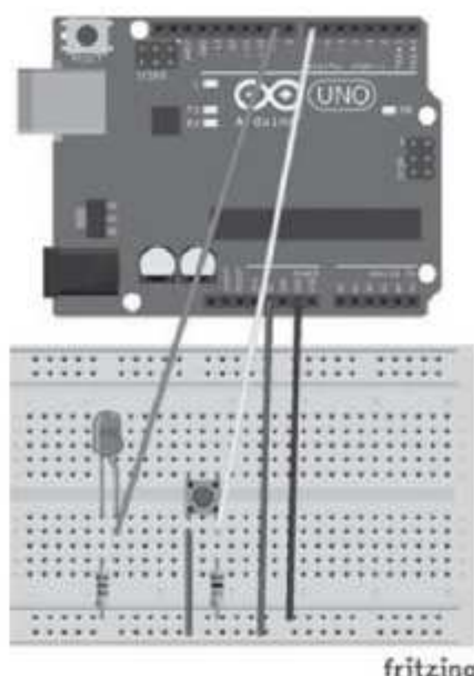


Figura 4.9 – Controle da luminosidade de um LED.

Conforme ilustra a Figura 4.9:

a) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
c) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra linha da protoboard.
d) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor de 220 ohms (ou 330 ohms).
e) Conecte o ânodo do LED ao pino 9 do Arduino.
f) Coloque o resistor de 10 kohms entre a linha de alimentação negativa e qualquer outra linha da protoboard.
g) Conecte uma das extremidades do botão ao resistor de 10 kohms.
h) Conecte a outra extremidade do botão ao pino 7.



Programa

No ambiente de desenvolvimento do Arduino, digite o sketch a seguir:

```
int LED = 9; // Pino no qual o LED está conectado
int BOTAO = 7; // Pino no qual o Botão está conectado
int valor = LOW;
int valorAnterior = LOW;
int estado = 0; // 0 = LED apagado, 1 = LED aceso
int brilho = 128;
unsigned long inicio;

void setup() {
pinMode(LED, OUTPUT); // Definir o pino como saída
  pinMode(BOTAO, INPUT); // Definir o pino com entrada
}

void loop() {
 valor = digitalRead(BOTAO);
  if ((valor == HIGH) && (valorAnterior == LOW)) {
estado = 1 - estado;
    inicio = millis(); // Obtém a quantidade de milissegundos
                       // após o Arduino ser inicializado
    delay (10);
  }
```

```
  // Verifica se o botão está sendo segurado pressionado
if ((valor == HIGH) && (valorAnterior == HIGH)) {
// Verifica se o botão está pressionado por mais
    // de 0,5 segundos
    if (estado == 1 && (millis() - inicio) > 500) {
 brilho++;
      delay(10);

      if (brilho > 255)
        brilho = 0;
    }
  }

  valorAnterior = valor;

  if (estado == 1)
    analogWrite(LED, brilho); // Define o nível de luminosidade
else
    analogWrite(LED, 0); // Apaga o LED
}
```

Como podemos observar no trecho de programa a seguir, utilizamos a função digitalRead para obter o estado do botão. Em seguida, verificamos se esse estado mudou para HIGH em relação ao valor obtido anteriormente. Caso isso tenha ocorrido, alternamos o estado do LED e também armazenamos o momento em que ocorreu o pressionamento.

```
valor = digitalRead(BOTAO);
if ((valor == HIGH) && (valorAnterior == LOW)) {
 estado = 1 - estado;
  inicio = milis();  // Obtém a quantidade de milissegundos
                     // após o Arduino ser inicializado
  delay (10);
}
```

Posteriormente, como podemos observar no trecho de código-fonte mostrado a seguir, verificamos se o botão permanece pressionado por mais de 0,5 segundo. Nesse caso, incrementamos a variável responsável por determinar o brilho (intensidade luminosa) do LED.

```
if ((valor == HIGH) && (valorAnterior == HIGH)) {
  // Verifica se o botão está pressionado por mais
  // de 0,5 segundo
  if (estado == 1 && (milis() - inicio) > 500) {
      brilho++;
      delay(10);
```

```
        if (brilho > 255)
          brilho = 0;
  }
}
```

Concluindo este exemplo, o sketch desenvolvido irá acender o LED com o brilho determinado, ou, caso contrário, apagará o LED, conforme podemos observar no trecho de programa a seguir.

```
if (estado == 1)
  analogWrite(LED, brilho); // Define o nível de luminosidade
else
  analogWrite(LED, 0); // Apaga o LED
```



PROJETO Nº 8 Uso de um LED RGB

Os LEDs RGB consistem em três LEDs encapsulados em um único invólucro. Dessa forma, podem ser utilizados para produzir uma grande combinação de cores e tonalidades. Neste projeto, vamos demonstrar a utilização desse tipo de LED, que, em conjunto com a função para gerar números aleatórios, irá produzir o acendimento do LED nas mais diversas cores.



Figura 4.10 – LED RGB.

Na Figura 4.10 podemos ver a aparência típica de um LED RGB, que possui quatro pinos apresentando as respectivas funções:

Pino	Nome	Função
1	R	LED vermelho
2	GND ou VCC	Alimentação
3	G	LED verde
4	B	LED azul

Observe que o pino 2 deverá ser conectado ao polo negativo quando o LED for do tipo cátodo comum ou ao polo positivo quando o LED RGD for do tipo ânodo comum. No projeto que montaremos a seguir, iremos considerar que o LED possui cátodo comum.



Material necessário

- 1 Arduino;
- 3 resistores de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED RGB;
- 1 protoboard;
- jumper cable.



Montagem do circuito

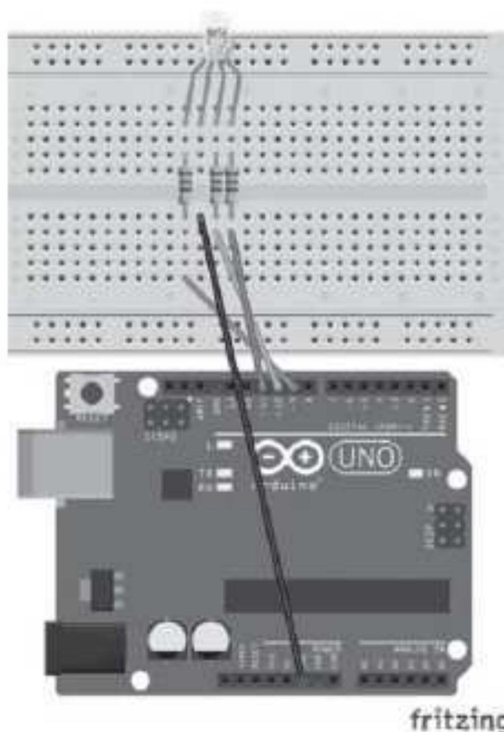


Figura 4.11 – Projeto com LED RGB.

Conforme ilustra a Figura 4.11:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Conecte o pino 2 (GND) do LED RGB à linha de alimentação negativa (preta) da protoboard.
c) Coloque um resistor de 220 ohms (ou 330 ohms) no pino 1 (Red – Vermelho) do LED RGB.
d) Conecte o outro pino desse resistor ao pino 9 do Arduino.

e) Coloque um resistor de 220 ohms (ou 330 ohms) no pino 3 (Green – Verde) do LED RGB.

f) Conecte o outro pino desse resistor ao pino 10 do Arduino.

g) Coloque um resistor de 220 ohms (ou 330 ohms) no pino 4 (Blue – Azul) do LED RGB.

h) Conecte o outro pino desse resistor ao pino 11 do Arduino.



Programa

Abra o ambiente de desenvolvimento do Arduino e digite o sketch a seguir:

```
int VERMELHO = 9;
int VERDE = 10;
int AZUL = 11;

int iVermelho = 0;
int iVerde = 0;
int iAzul = 0;

void setup()
{ pinMode (VERMELHO, OUTPUT);
  pinMode (VERDE, OUTPUT);
  pinMode (AZUL, OUTPUT);
}

void loop()
{ analogWrite(VERMELHO, iVermelho);
  analogWrite(VERDE, iVerde);
  analogWrite(AZUL, iAzul);

  delay(500);

  iVermelho = random(256);
  iVerde = random(256);
  iAzul = random(256);
}
```

O conceito básico deste sketch é utilizar a função random para gerar um número aleatório entre 0 e 255, que correspondem aos que podem ser enviados para uma saída PWM do Arduino. Dessa forma, sorteamos um valor para cada um dos componentes de cores do LED, ou seja:

```
iVermelho = random(256);
iVerde = random(256);
iAzul = random(256);
```

Em seguida, o valor sorteado é colocado na respectiva saída PWM, produzindo uma variação na tonalidade de cor de um dos componentes do LED RGB.



Programa

No ambiente de desenvolvimento do Arduino, digite o seguinte programa:

```
int VERMELHO = 9;
int VERDE = 10;
int AZUL = 11;

int iVermelho = 0;
int iVerde = 0;
int iAzul = 0;
unsigned long int tempo = 0;

void setup()
{ pinMode (VERMELHO, OUTPUT);
  pinMode (VERDE, OUTPUT);
pinMode (AZUL, OUTPUT);
}

void loop()
{ analogWrite(VERMELHO, iVermelho);
  analogWrite(VERDE, iVerde);
  analogWrite(AZUL, iAzul);
  delay(100);
  iVermelho = sin(tempo*tempo/50000.0)*128+128;
  iVerde = sin(tempo*tempo/100000.0)*128+128;
  iAzul = sin(tempo*tempo/70000.0)*128+128;
  tempo += 10;
}
```

Este programa é bastante similar ao anterior. A diferença básica é que, enquanto no outro programa utilizamos a função random para gerar valores aleatórios para cada uma das cores, agora iremos realizar a definição do valor que define a intensidade luminosa de cada cor com base na valor obtido pelo cálculo do seno. Dessa forma, observe o uso da função sin no código-fonte anterior.

Exercícios

1. Considerando o diagrama mostrado na Figura 4.12, desenvolva um sketch que a cada pressionamento do botão troque a cor do LED RGB, ou seja, inicialmente o LED deve estar apagado, ao pressionar o botão, a cor vermelha deve acender, após outro pressionamento a cor verde acende e um novo pressionamento deverá mostrar a cor azul.

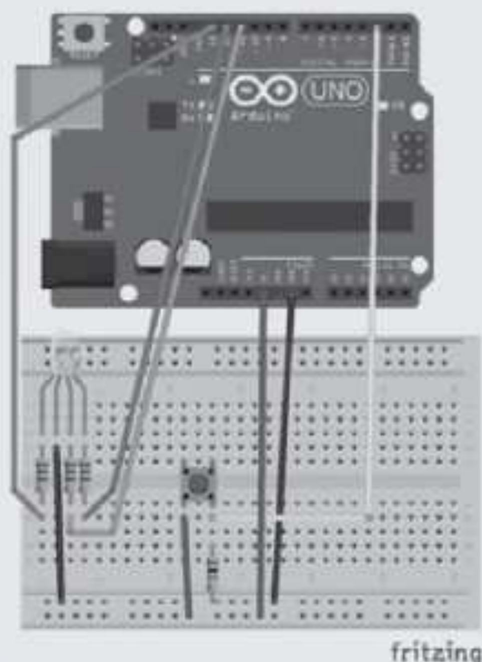


Figura 4.12 – Circuito proposto para o Exercício 1.

2. Altere o circuito da Figura 4.12 para trabalhar com as saídas PWM do Arduino e acrescente um potenciômetro no pino analógico A0. Em seguida, altere o sketch de modo que o potenciômetro determine a intensidade luminosa da cor do LED RGB que está acesa.

3. Altere o circuito da Figura 4.12 para utilizar três botões, cada um dos quais conectado a uma saída digital do Arduino. Um botão deverá acender e apagar a cor vermelha do LED RGB, o outro deverá acender e apagar a cor verde, e o terceiro botão deverá acender e apagar a cor azul.

4. Reproduza um cenário similar ao de um semáforo de carros e pedestres. Supondo o estado inicial do cenário com semáforo de carros (SC) sendo vermelho (PARE) e o semáforo de pedestres (SP) sendo verde (SIGA), deve-se programar a sequência de luzes indicando os estados do SC sincronizado com os de SP. Algumas especificações a serem seguidas:
   - Os sinais vermelho e verde de SC têm duração de 10 segundos cada.
   - O sinal amarelo de SC tem duração de 2 segundos.
   - O sinal vermelho de SP ficará aceso durante todo o tempo em que o vermelho e o amarelo de SC estiverem acesos, impedindo a passagem de pedestres enquanto os carros puderem transitar.
   - O sinal verde de SP ficará aceso durante todo o tempo em que o vermelho de SC ficar aceso, indicando que os pedestres estão livres para atravessar.
   - Antes da transição do sinal verde para o vermelho de SP, faltando 2 segundos para a transição, o sinal verde pisca rapidamente 2 vezes, indicando aos pedestres que se tornará vermelho.



Material necessário

- 1 Arduino;
- 5 resistores de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 5 LEDs (2 vermelhos, 2 verdes e 1 amarelo);
- 1 protoboard;
- jumper cable.



Montagem do circuito

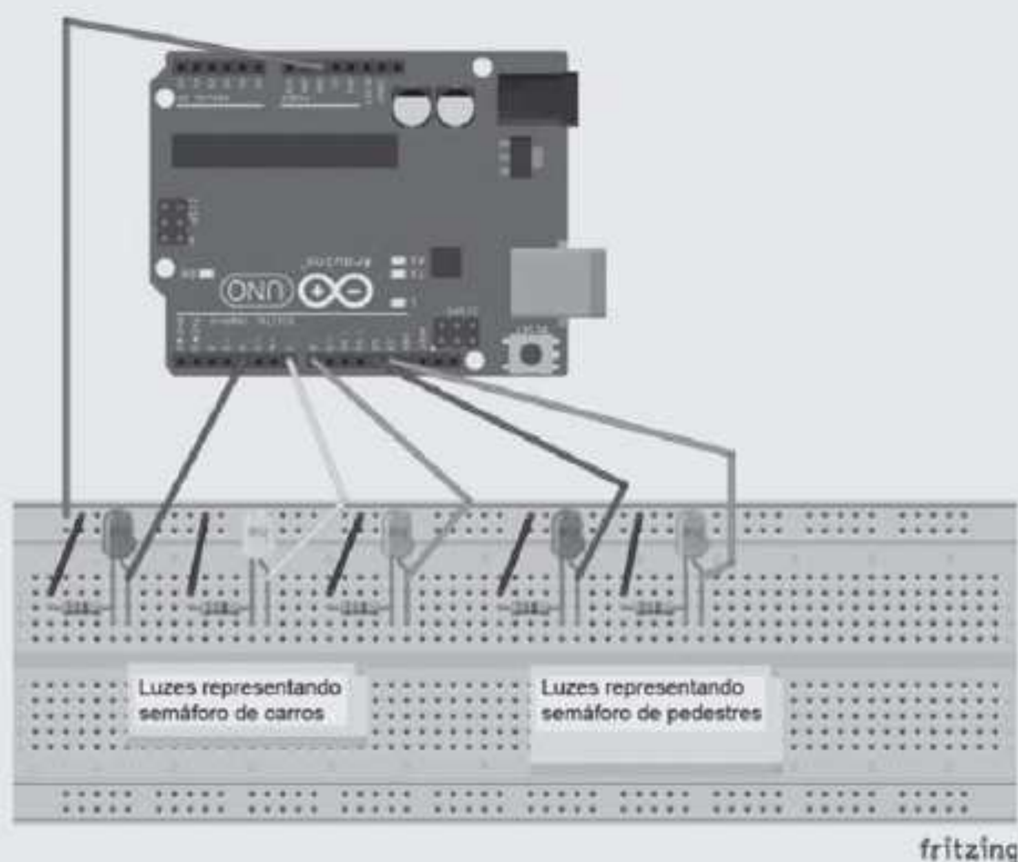


Figura 4.13 – Projeto do semáforo completo (para versão colorida, consulte o site da Editora Érica).

Conforme ilustra a Figura 4.13:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (azul) da protoboard.

b) Coloque 5 resistores de 220 ou 330 ohms; entre com uma ligação na linha de alimentação negativa e qualquer outra da protoboard.

c) Coloque 5 LEDs com o cátodo (lado chanfrado) conectado em cada um dos resistores.

d) Conecte o ânodo dos LEDs na seguinte ordem:

- no pino 4 do Arduino, um LED vermelho (vermelho de SC);
- no pino 7 do Arduino, um LED amarelo (amarelo de SC);
- no pino 8 do Arduino, um LED verde (verde de SC);
- no pino 12 do Arduino, um LED vermelho (vermelho de SP);
- no pino 13 do Arduino, um LED verde (verde de SP), rapidamente 2 vezes, indicando aos pedestres que se tornará vermelho.

5

Porta Serial

O Arduino possui capacidade de comunicação serial. Dessa forma, é possível enviar e receber dados em um sketch. Assim, os projetos criados podem interagir com outros dispositivos de hardware. Este capítulo traz dois projetos que demonstrarão esses conceitos, ou seja, envio e recebimentos de dados pela porta serial, os quais serão também amplamente utilizados nos capítulos seguintes.

5.1 Saída serial



PROJETO Nº 9
Enviando dados para a saída serial

O objetivo deste projeto é enviar um valor, colocado em uma entrada analógica por meio de um potenciômetro, pela porta de comunicação serial do Arduino. Em seguida, demonstraremos como utilizar a opção Monitor Serial, disponível no ambiente de desenvolvimento do Arduino, para visualizar os valores enviados.



Material necessário

- 1 Arduino;
- 1 potenciômetro;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

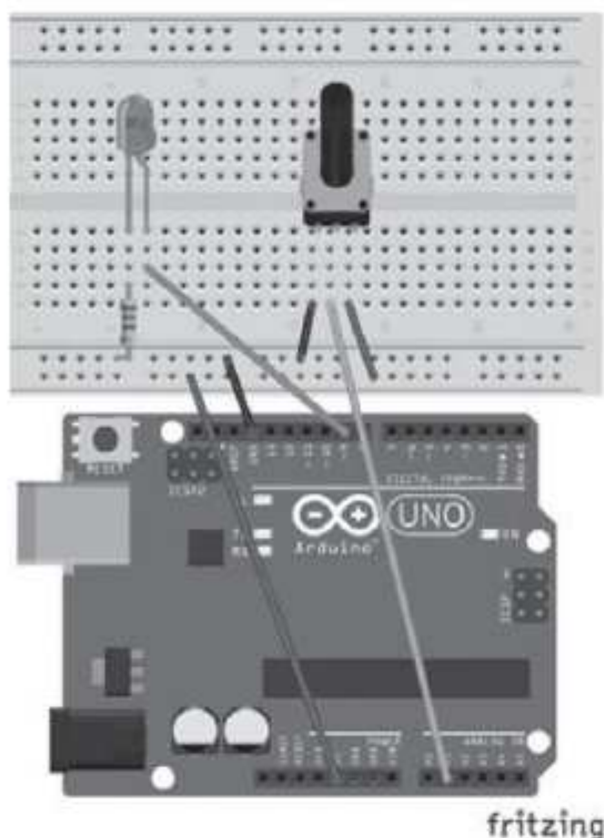


Figura 5.1 – Exemplo de utilização da saída serial.

Conforme ilustra a Figura 5.1, realize os seguintes passos para montar o circuito que será utilizado no projeto:

a) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
c) Conecte um LED utilizando um resistor de 220 ohms (ou 330 ohms).
d) Conecte o LED no pino analógico 9.
e) Conecte o potenciômetro na protoboard conforme a Figura 5.1 (botão de girar virado para você).
f) Conecte o pino da esquerda do potenciômetro na linha de alimentação GND.
g) Conecte o pino da direita do potenciômetro na linha de alimentação positiva.
h) Conecte o pino do centro do potenciômetro no pino analógico A1 do Arduino.



Programa

Após montar o projeto, vá até o ambiente de desenvolvimento do Arduino e digite o programa a seguir:

```
// Controla a intensidade luminosa de um LED através do
// uso de um potenciômetro conectado a uma entrada analógica

// Pino analógico do Arduino ao qual será ligado o potenciômetro
int POT = A1;

// Pino do Arduino ao qual será conectado o LED
int LED = 9;

int valor = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  valor = analogRead(POT);
  if(valor > 0) {
    //Acende o LED com intensidade proporcional ao valor obtido
    analogWrite(LED, (valor/4));

    // Exibe no Serial Monitor o valor obtido do potenciômetro
    Serial.println(valor);
  }
}
```

Na setup do sketch devemos inicializar a porta serial e estabelecer a velocidade de comunicação pelo comando Serial.begin(9600), em que 9.600 é a velocidade em que ocorrerá a transmissão dos dados. Para enviar dados, devemos utilizar Serial.println(valor), em que valor representa o dado a ser transmitido e o println acrescentará uma quebra de linha (CR+LF) após o dado.

Conforme podemos observar na Figura 5.2, clique no ícone indicado para abrir a janela do Monitor Serial.

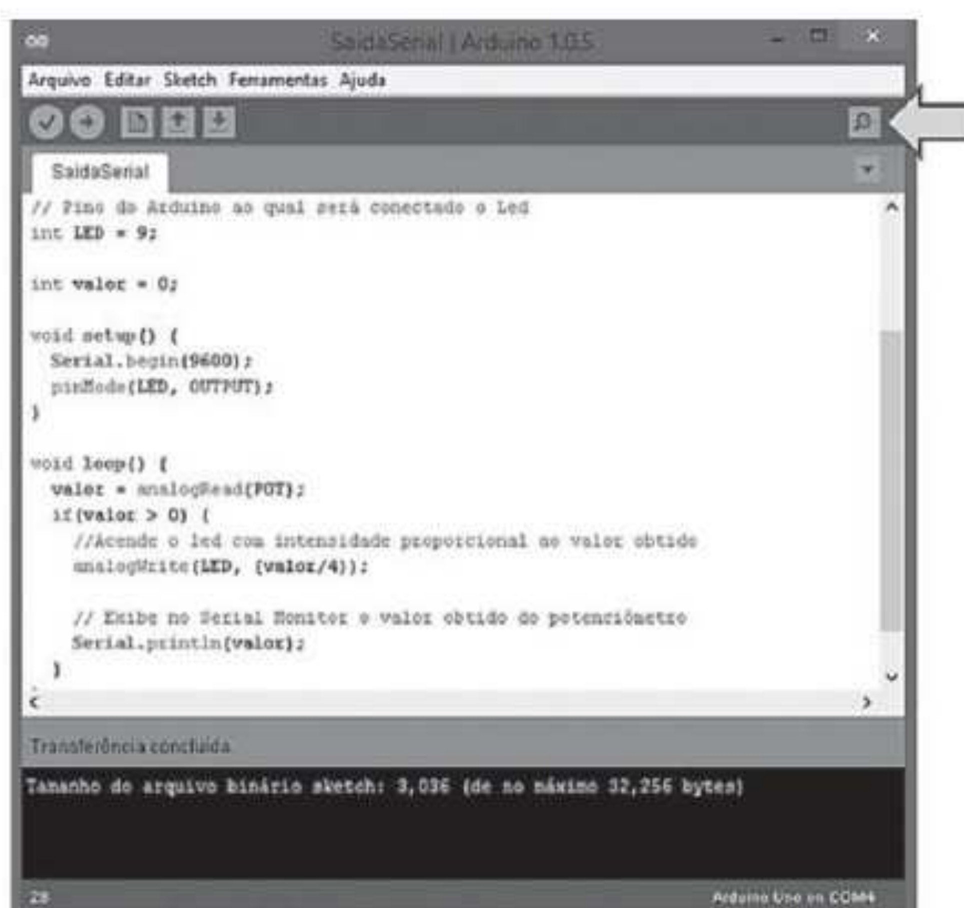


Figura 5.2 – Botão para abrir o Monitor Serial.

A janela do Monitor Serial (Figura 5.3) será exibida e apresentará os dados que são enviados pelo Arduino, neste exemplo, os valores que estão na porta analógica A1.

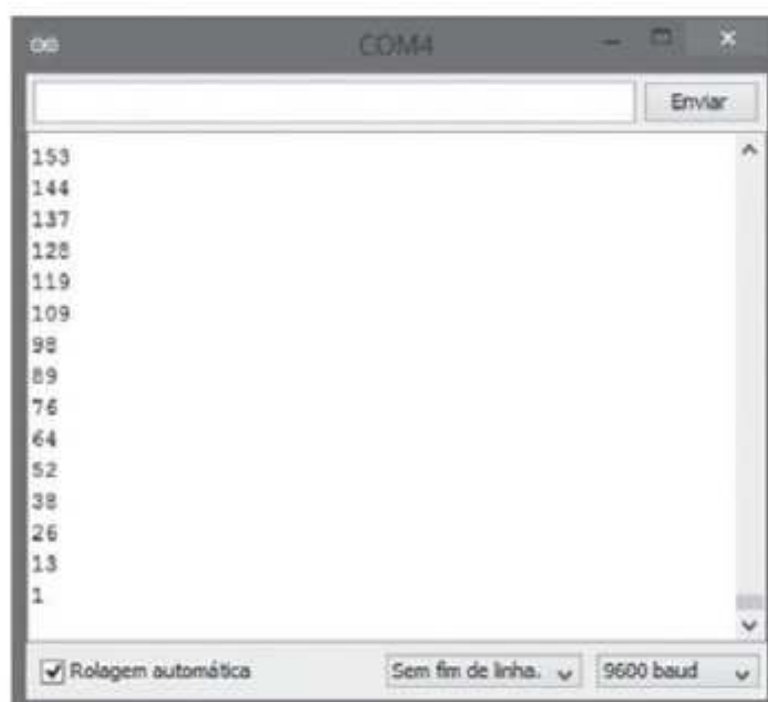


Figura 5.3 – Janela do Monitor Serial.



Programa

Agora vamos realizar uma pequena alteração no sketch de modo a mostrar os dados enviados, separados por uma vírgula e sem a quebra de linha:

```
//Controla a intensidade luminosa de um LED através do
// uso de um potenciômetro conectado a uma entrada analógica do
// Arduino

// Pino analógico do Arduino ao qual será ligado o potenciômetro
int POT = A1;

// Pino do Arduino ao qual será conectado o LED
int LED = 9;

int valor = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  valor = analogRead(POT);
  if(valor > 0) {
    //Acende o LED com intensidade proporcional ao valor obtido
    analogWrite(LED, (valor/4));

    // Exibe no Serial Monitor o valor obtido do potenciômetro
    Serial.print(valor);
    Serial.print(", ");
  }
}
```

Neste caso, observe na Figura 5.4 o uso do comando Serial.print que realiza o envio dos dados, sem porém acrescentar a quebra de linha entre os valores.



Figura 5.4 – Envio de dados para a saída serial.

Tanto Serial.print quanto Serial.println podem receber um parâmetro opcional que possibilitará formatar o valor para ser enviado pela porta de comunicação serial. Dessa forma, temos que:

- Serial.print(64, BIN) apresentará o valor em binário, neste exemplo, "1000000".
- Serial.print(64, DEC) realizará o envio em decimal, ou seja, "64".
- Serial.print(64, HEX) mostrará o valor em hexadecimal, ou seja, "40".
- Serial.println(9.87654, 0) estabelecerá apenas a parte inteira do número, isto é, "9".
- Serial.println(9.87654, 2) indicará o número real com duas casas decimais, neste exemplo, "9.87".
- Serial.println(9.87654, 4) apresentará o número real com quatro casas decimais, neste exemplo, "9.8765".

5.2 Entrada serial



PROJETO Nº 10
Controle de um LED a partir da entrada serial

Além de enviar dados pela porta serial, o Arduino também poderá recebê-los. Neste próximo projeto vamos ilustrar esse conceito. A partir de um dado recebido pela entrada serial, vamos determinar se o LED deverá ser ligado ou desligado.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

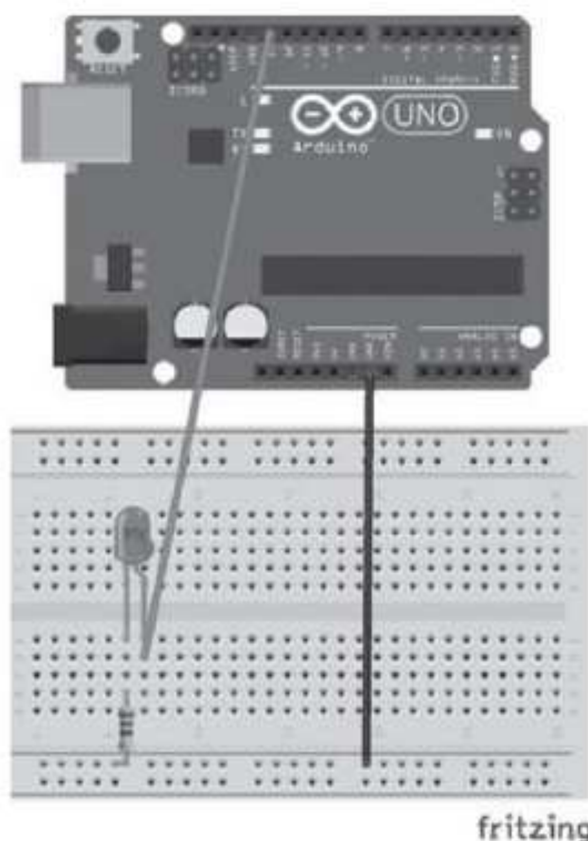


Figura 5.5 – Projeto que ilustra o uso da entrada serial.

Conforme ilustra a Figura 5.5:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra linha da protoboard.
c) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
d) Conecte o ânodo do LED ao pino 13 do Arduino.



Programa

Inicie o ambiente de desenvolvimento (IDE) do Arduino e digite o sketch (programa) a seguir:

```
// Exemplo de entrada de dados serial

int LED = 13;
int entrada = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  Serial.println("Digite 1 para acender o LED ou 0 para apagar:");
}

void loop() {
  if (Serial.available() > 0) {
    // Ler o byte que está na entrada serial
    entrada = Serial.read();

    // Informar o valor que foi recebido
    Serial.print("Recebido: ");
    Serial.println(entrada, DEC);

    // Acender ou apagar o LED de acordo com o
    // valor recebido na entrada serial
    if (entrada == '0')
      digitalWrite(LED, LOW);
    else if (entrada == '1')
      digitalWrite(LED, HIGH);
    else
      Serial.println("Por favor, digite apenas 0 ou 1!");
  }
}
```

Na função setup, devemos inicializar e definir a velocidade de comunicação da porta serial. Em seguida, no laço do programa, vamos utilizar a função Serial.available para determinar se há bytes na entrada serial aguardando para serem lidos. Dessa maneira, quando o retorno da função Serial.available for um valor maior que zero, devemos usar a função Serial.read para obtermos os dados.

Quando o dado recebido for o caractere '0', o LED deverá ser apagado; quando recebermos o caractere '1', o LED será aceso. Para qualquer outro valor recebido, enviaremos pela saída serial uma mensagem informando ao usuário para digitar apenas zero ou um.



Figura 5.6 – Colocando dados na entrada serial.

Conforme ilustrado na Figura 5.6, o valor que é digitado no Monitor Serial é tratado como um caractere ASCII. Dessa forma, quando inserimos o valor um, temos o caractere '1' cujo valor numérico é 49. Uma pequena alteração no código-fonte do nosso sketch permitirá mostrarmos isso:

```
// Informar o valor que foi recebido
Serial.print("Recebido - Valor com caractere: ");
Serial.print((char) entrada);
Serial.print(", valor numérico: ");
Serial.println(entrada, DEC);
```

Conforme podemos observar na Figura 5.7, ao tratarmos o dado de entrada como char, o programa mostrará o valor como um dado caractere, ou seja, '1'. Por outro lado, quando usamos a impressão pela opção de formatação DEC, teremos o seu valor numérico inteiro, ou seja, 49.



Figura 5.7 – Exibição formatada do caractere recebido.

Exercícios

1. Considerando o circuito mostrado na Figura 5.8, escreva um sketch que receba um caractere pela entrada serial. Se o caractere for 'A', a intensidade luminosa do LED deve ser aumentada. Quando o caractere for 'D', a intensidade luminosa deve ser reduzida, e quando o caractere digitado for 'R', o LED deverá ser apagado.

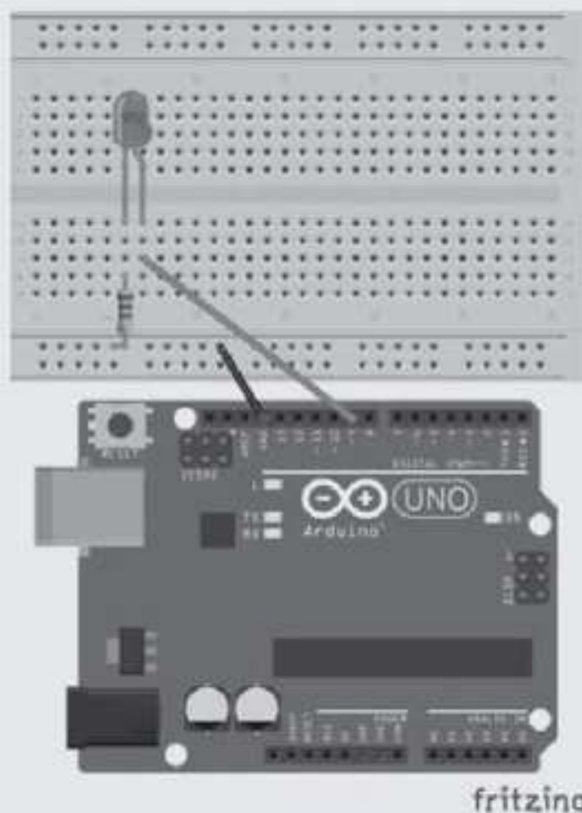


Figura 5.8 – Led conectado a uma saída PWM.

2. Altere o sketch desenvolvido no exercício anterior para mostrar na saída serial o valor da intensidade luminosa do LED.

6

Produzindo Som

Além da possibilidade de mostrarmos uma informação de modo visual, ou seja, nos exemplos anteriores usamos LEDs como atuadores, também podemos reproduzir sinais sonoros. Desta forma, podemos ter nos projetos a reprodução de indicações sonoras, ou mesmo integrar as duas formas, ou seja, indicações sonoras e visuais. Neste capítulo, abordamos a geração de som por meio do Arduino e também o processo contrário, ou seja, a conversão de uma vibração em sinais elétricos.

6.1 Buzzer



PROJETO Nº 11
Utilização do buzzer

O objetivo deste projeto é acionar um buzzer utilizando as funções tone e noTone. O buzzer é um dispositivo que permite a geração de sinais sonoros (bipes) como aqueles encontrados em computadores. Para produzir o som, um buzzer vibra por meio de um oscilador. Essa oscilação é determinada por uma frequência que, por sua vez, define um som específico.



Material necessário

- 1 Arduino;
- 1 buzzer;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

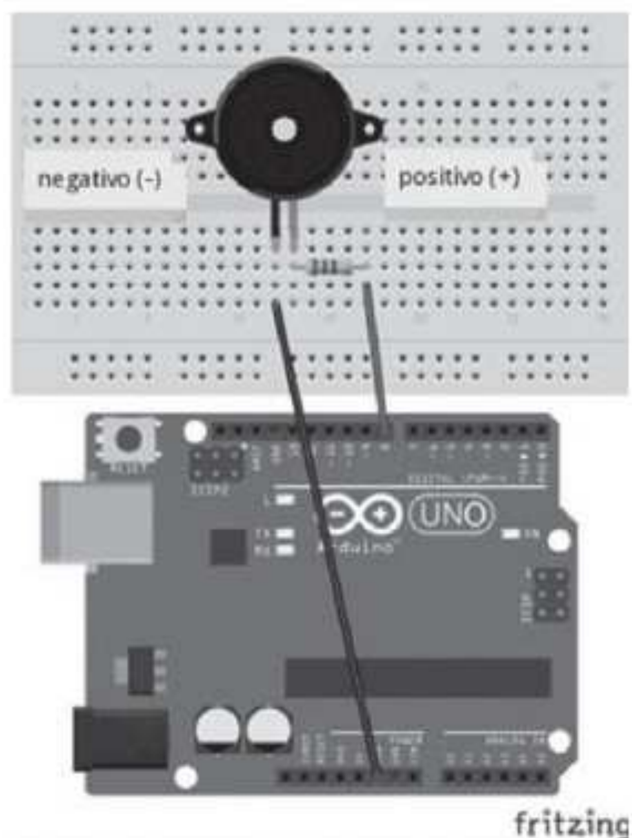


Figura 6.1 – Uso do buzzer.

Conforme podemos observar na Figura 6.1:

a) Conecte o pino GND do Arduino ao negativo do buzzer colocado na protoboard.
b) Coloque o resistor de 220 ohms (ou 330 ohms) ligado ao pino positivo do buzzer.
c) Nesse resistor, conecte um jumper até a porta digital 8 do Arduino.



Programa

Após montar o circuito que será utilizado neste projeto, digite o sketch a seguir:

```
int BUZZER = 8; // Pino ao qual o buzzer está conectado

void setup(){
  pinMode(BUZZER, OUTPUT);
}

void loop(){
  tone(BUZZER, 1200); // Define pino e a frequência
  delay(500);
```

```
  noTone(BUZZER); // Desliga o buzzer
  delay(500);
}
```

Na função setup definimos o pino no qual o buzzer está conectado como saída e, em seguida, na função loop ativamos o buzzer através da função tone, que recebe como parâmetros o pino e a frequência do sinal sonoro que será gerado. O valor da frequência é definido em hertz. A função noTone parará o som emitido pelo buzzer.



Programa

Agora vamos desenvolver outro exemplo que demonstra o uso do buzzer. Para isso, digite o sketch a seguir:

```
int BUZZER = 8; // Pino ao qual o buzzer está conectado

void setup(){
  pinMode(BUZZER, OUTPUT);
}

void loop(){
  tone(BUZZER, 1200, 500); // Define pino, a frequência e duração
  delay(1000);
}
```

Podemos observar no programa que a função tone pode receber um terceiro parâmetro, que consiste na duração deve ser um valor inteiro que expressa o tempo em milissegundos. Também podemos notar que, quando utilizamos essa sintaxe da função tone, torna-se desnecessário utilizar a função noTone para finalizar o som emitido pelo buzzer.



Programa

Digite o seguinte sketch:

```
int BUZZER= 8; // Pino ao qual o buzzer está conectado

int numNotas = 10;

// Vetor de inteiros contendo diversas frequências
int notas[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440};
// Notas         C   C#   D    D#   E    F    F#   G    G#   A
```

```
/*
 * C=Dó D=Ré E=Mi F=Fá G=Sol A=Lá B=Si
 *
 * As notas sem o "#" são as notas naturais (fundamentais).
 * Aquelas com o "#" são chamadas "notas sustenidas", por exemplo:
* C# equivale ao "Dó Sustenido".
*/

void setup(){
  pinMode(BUZZER, OUTPUT);
}

void loop(){
  for (int i = 0; i < numNotas; i++){
    tone(BUZZER, notas[i]);
    delay(500);
  }
  noTone(BUZZER);
}
```

Neste exemplo usamos o laço de repetição for para poder acessar cada elemento do vetor declarado, emitindo, dessa forma, a respectiva frequência pelo buzzer.

6.2 Cristal piezoelétrico

Um cristal piezoelétrico possui a capacidade de converter uma oscilação mecânica (pressão sobre ele) em um sinal elétrico. O inverso desse princípio, ou seja, converter um sinal elétrico em oscilação mecânica, é exatamente o que ocorre em um buzzer. A pressão sobre um cristal piezoelétrico gera uma pequena corrente elétrica, e, por esse motivo, eles possuem polaridade. Como podemos notar na Figura 6.2, o centro prateado consiste no polo positivo que está conectado a um fio vermelho, enquanto a borda dourada é o polo negativo, que se encontra ligada ao fio preto.



Figura 6.2 – Cristal piezoelétrico (para versão colorida, consulte o site da Editora Érica).

Quando o cristal piezoelétrico é pressionado, a corrente elétrica atinge um limiar, que depende da pressão. A partir desse valor de limiar é que verificamos se houve ou não um toque nele. Esse limiar deve ser ajustado para definir a sensibilidade necessária do toque, e um valor analógico será lido pelo Arduino.



PROJETO Nº 12
Utilização do cristal piezoelétrico

O objetivo deste projeto é utilizar um cristal piezoelétrico para identificar um toque, ou seja, uma pressão realizada sobre o componente e, com isso, acionar um buzzer.



Material necessário

- 1 Arduino;
- 1 buzzer;
- 1 cristal piezoelétrico;
- 1 resistor de 1 kohm (marrom, preto, vermelho);
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

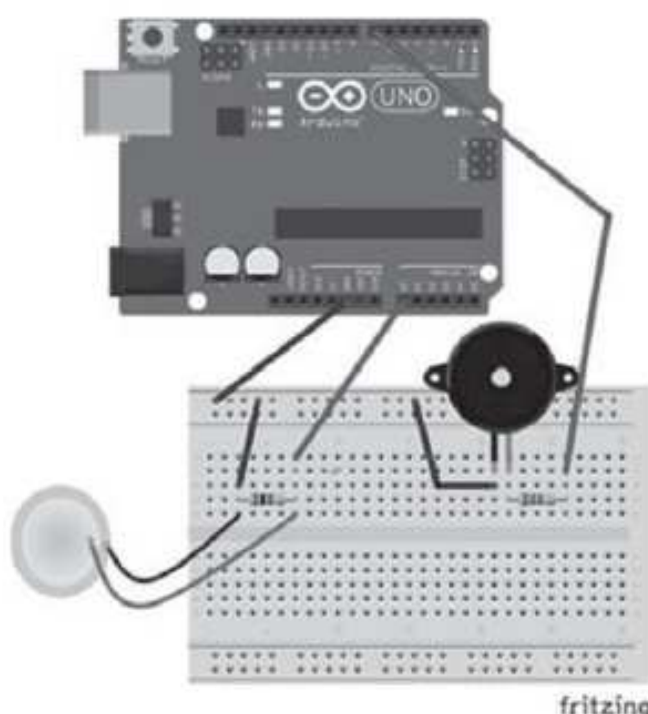


Figura 6.3 – Uso do cristal piezoelétrico.

Conforme podemos observar na Figura 6.3, realize os passos a seguir para montar o projeto:

a) Conecte o GND do Arduino na alimentação GND da protoboard.

b) Conecte um resistor de 1 kohm no sentido horizontal da protoboard.

c) Conecte os fios do cristal piezoelétrico nas extremidades desse resistor.
d) Conecte um jumper no GND da protoboard e na linha do negativo do piezo.
e) Conecte um jumper no positivo do piezo e no pino A0 do Arduino.
f) Coloque o buzzer no sentido horizontal da protoboard.
g) Conecte o pino GND do Arduino ao negativo do buzzer.
h) Coloque o resistor de 330 ohms ligado ao positivo do buzzer.
i) Nesse resistor, conecte um jumper até a porta digital 7 do Arduino.



Programa

Após montar o circuito que será utilizado neste projeto, vá até o ambiente de desenvolvimento do Arduino e digite o sketch a seguir:

```
int BUZZER = 7 ;
int PIEZO = A0;

// Valor do limiar, ou seja, define se houve toque ou não
int limiar = 50;

// Armazena o valor que será gerado pelo cristal piezoelétrico
int valor_piezo;

void setup() {
  pinMode(BUZZER, OUTPUT);
  pinMode(PIEZO, INPUT);
  Serial.begin(9600);
}

void loop() {
  valor_piezo = analogRead(PIEZO);

  // Verifica se o valor lido pela pressão no piezo
  // ultrapassa o limiar
  if (valor_piezo>= limiar) {
    // Aciona o buzzer
    tone(BUZZER, 1600, 200);

    // Exibe no Monitor Serial a palavra "Bip!", indicando que
    // foi identificado o toque no cristal piezoelétrico
    Serial.println("Bip!");
  }
  delay(100);
}
```

Nesse sketch podemos observar que o cristal piezoelétrico está conectado à entrada analógica zero (A0) do Arduino. Em seguida, na função loop, ficamos realizando a leitura dos valores gerados pelo cristal piezoelétrico, ou seja:

```
valor_piezo = analogRead(PIEZO);
```

O valor obtido é verificado em relação àquele definido para a variável limiar, e, quando superá-lo, o buzzer será acionado.

Exercícios

1. Adicione um potenciômetro ao Projeto nº 11 e modifique o programa, de modo que ele controle o intervalo de toque do buzzer por meio da variação do potenciômetro, ou seja, de 0 até 1.023 ms (milissegundos). Dica: Utilize a função tone.
2. Altere o projeto mostrado na Figura 6.3 trocando o buzzer por um LED. Quando o valor de limiar for atingido, o LED deverá ser aceso, caso contrário deverá permanecer apagado.
3. Desenvolva um pequeno teclado musical. Conforme ilustra a Figura 6.4, na protoboard conecte sete botões, que serão equivalentes às sete notas musicais (C=Dó, D=Ré, E=Mi, F=Fá, G=Sol, A=Lá e B=Si). Cada vez que um botão for pressionado, a nota correspondente deverá ser reproduzida no buzzer.

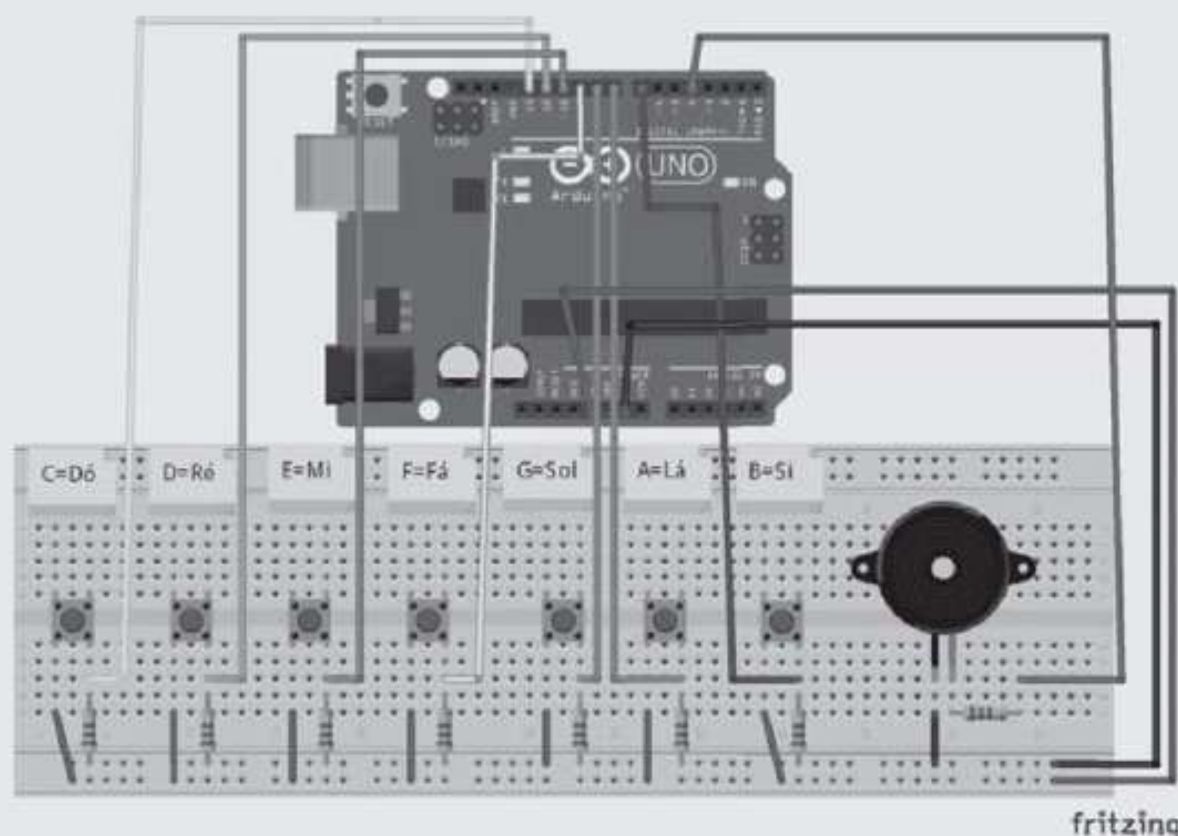


Figura 6.4 – Teclado musical.

7 Displays

Existem diversas maneiras de apresentarmos para um usuário as informações alfanuméricas geradas pelos nossos projetos. Neste capítulo abordaremos as três maneiras mais empregadas: o uso do Display de Cristal Líquido (LCD), dos Displays de LEDs com sete segmentos e das Matrizes de LEDs.

7.1 Liquid Crystal Display (LCD)



PROJETO Nº 13
Utilização do display de LCD de 16x2

O objetivo deste projeto é demonstrar a utilização de um display de cristal líquido (LCD) por meio da biblioteca LiquidCrystal.h, que apresenta várias funcionalidades que auxiliarão na configuração e no tratamento dos dados enviados ao LCD. Iremos utilizar um modelo de display de LCD bastante comum e amplamente utilizado, que é o de duas linhas e 16 caracteres, porém os conceitos empregados podem ser aplicados em outros tipos de displays de LCD com configurações diferentes, como quatro linhas por 16 caracteres, entre outros.

Inicialmente, é importante identificar a função de cada um dos pinos do display. Na Figura 7.1 é mostrada a ordem dos pinos de 1 até 16.

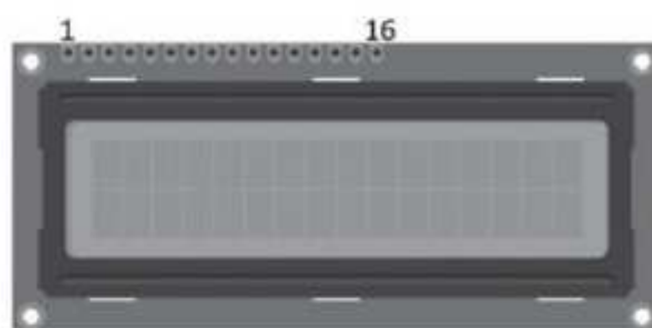


Figura 7.1 – Display de cristal líquido (LCD).

Na tabela a seguir temos a função de cada um dos pinos do display de LCD Modelo ACM 1602K de 16 linhas por duas colunas ou modelos similares.

Pino	Nome	Função
1	Gnd	Alimentação (polo negativo)
2	Vcc	Alimentação 5 V (polo positivo)
3	V0	Ajuste do contraste
4	RS	Habilita ou desabilita a seleção de registrador
5	R/W	Leitura / Escrita
6	E	Habilita a escrita no LCD
7	DB0	Dado
8	DB1	Dado
9	DB2	Dado
10	DB3	Dado
11	DB4	Dado
12	DB5	Dado
13	DB6	Dado
14	DB7	Dado
15	BL+	Alimentação de 5 V da Backlight (polo positivo)
16	BL-	Alimentação Gnd da Backlight (polo negativo)



Material necessário

- 1 Arduino;
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

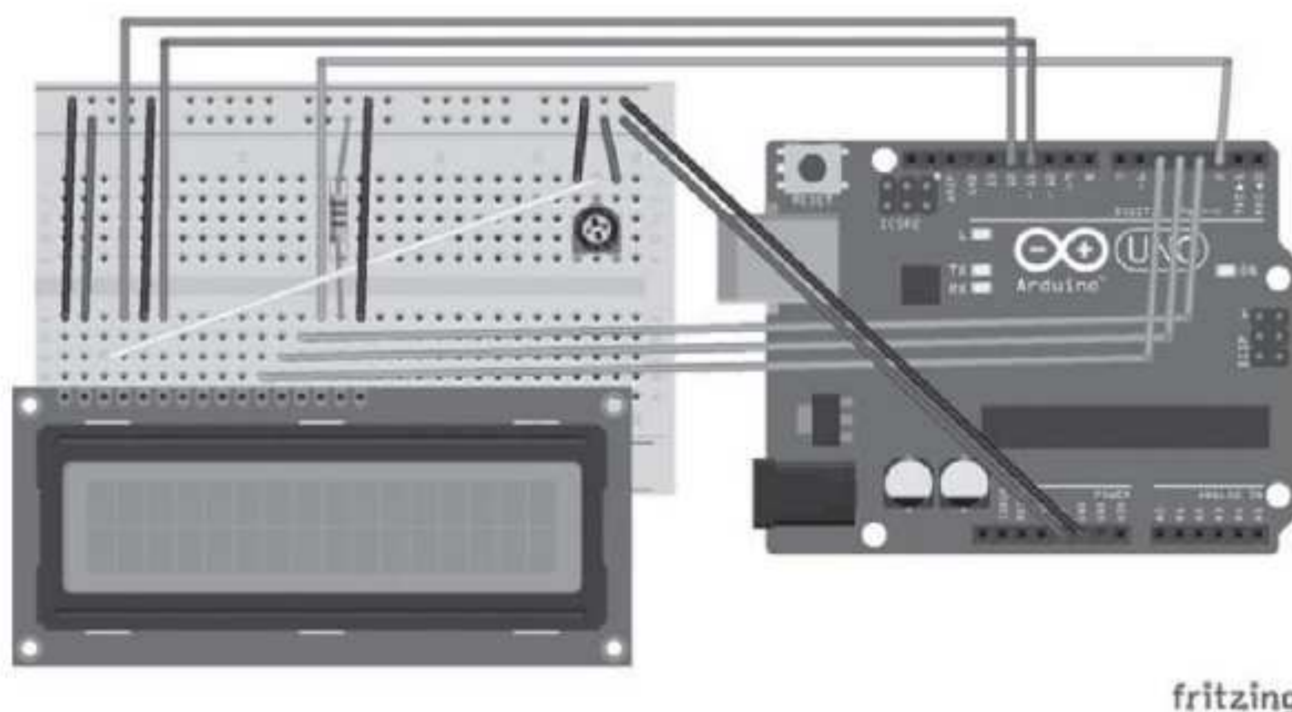


Figura 7.2 – Diagrama da conexão do LCD com o Arduino.

Considerando o esquema apresentado na Figura 7.2, realize os seguintes passos para montar o circuito que será utilizado neste projeto:

a) Pino 1 do LCD ligado ao GND do Arduino.
b) Pino 2 do LCD ligado ao 5 V do Arduino.
c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.



Programa

No ambiente de desenvolvimento do Arduino, digite o seguinte sketch:

```
#include <LiquidCrystal.h>

// Pinagem do ACM 1602K
// Pin Símbolo Função                  Conectar
//  1  Vss     Alimentação (-)         GND
//  2  Vdd     Alimentação (+5V)       VCC
//  3  Vo      Ajuste de contraste     Potenciômetro
//  4  RS      Seleção     Arduino     12
//  5  R/W     Leitura/Escrita         GND
//  6  E       Habilitar a escrita     Arduino 11
//  7  DB0     Bit de dados 0          NC
//  8  DB1     Bit de dados 1          NC
//  9  DB2     Bit de dados 2          NC
// 10  DB3     Bit de dados 3          NC
// 11  DB4     Bit de dados 4          Arduino 5
// 12  DB5     Bit de dados 5          Arduino 4
// 13  DB6     Bit de dados 6          Arduino 3
// 14  DB7     Bit de dados 7          Arduino 2
// +   BL+     Alimentação BL+ Resistor de 1 k para VCC
// -   BL-     Alimentação BL- GND

#define TEMPO_ATUALIZACAO 500

LiquidCrystal lcd (12, 11, 5, 4, 3, 2);

void setup() {
pinMode(12, OUTPUT);
pinMode(11, OUTPUT);

// Iniciar o LCD com dimensões 16x2(Colunas x Linhas)
lcd.begin (16, 2);
}

void loop() {
  // Apagar o conteúdo exibido no display de LCD
  lcd.clear();

// Posicionar o cursor na coordenada especificada
lcd.setCursor(0, 0);

// Exibir no LCD
  lcd.print("Arduino Descomplicado");

// Posicionar o cursor na coordenada especificada
lcd.setCursor(0, 1);
```

```
// Exibir no LCD
  lcd.print("Claudio e Humberto");

  delay(TEMPO_ATUALIZACAO);
}
```

Como podemos observar neste programa, devemos criar um objeto a partir da classe LiquidCrystal informando os pinos do Arduino que estarão conectados. O primeiro parâmetro é o pino do Arduino conectado ao seletor de registrador; em seguida, temos o pino que será responsável por habilitar a leitura e escrita. Os quatro parâmetros a seguir são os quatro pinos de dados que serão utilizados.

```
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
```

Em seguida, na função setup, inicializamos o display com a quantidade de linhas e colunas do dispositivo.

```
lcd.begin (16, 2);
```

Na função loop podemos utilizar lcd.setCursor para definir a coluna e a linha na qual a informação será exibida e lcd.print para exibir o conteúdo da cadeia de caractere, passada como parâmetro, no display de LCD, por exemplo:

```
lcd.setCursor(0, 0);
lcd.print("Arduino Descomplicado");
```



Programa

No ambiente de desenvolvimento do Arduino, implemente o programa a seguir:

```
#include <LiquidCrystal.h>

// Pinagem do ACM 1602 K
// Pin Símbolo Função                Conectar
//  1  Vss     Alimentação (-)       GND
//  2  Vdd     Alimentação (+5 V)     VCC
//  3  Vo      Ajuste de contraste   Potenciômetro
//  4  RS      Seleção     Arduino 12
//  5  R/W     Leitura/Escrita       GND
//  6  E       Habilitar a escrita   Arduino 11
//  7  DB0     Bit de dados 0        NC
```

```
// 8  DB1    Bit de dados 1      NC
// 9  DB2    Bit de dados 2      NC
// 10 DB3    Bit de dados 3      NC
// 11 DB4    Bit de dados 4      Arduino 5
// 12 DB5    Bit de dados 5      Arduino 4
// 13 DB6    Bit de dados 6      Arduino 3
// 14 DB7    Bit de dados 7      Arduino 2
// +  BL+    Alimentação BL+     Resistor de 1 k para VCC
// -  BL-    Alimentação BL-     GND

#define TEMPO_ATUALIZACAO 500

LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
int inicio = 0, tamanho = 1;
boolean alterar = false;

void setup() {
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);

  // Iniciar o LCD com dimensões 16x2(Colunas x Linhas)
  lcd.begin (16, 2);
}

void loop() {
  // Apagar o conteúdo exibido no display de LCD
  lcd.clear();

  String nome = "Arduino - Uma Abordagem Didática";
  if (tamanho < 16) {
    // Posicionar o cursor na coordenada especificada
    lcd.setCursor(16 - tamanho, 0);

    // Exibir no LCD
    lcd.print(nome.substring(inicio, tamanho));
    tamanho++;
  }
  else {
    if (!alterar){
      alterar = !alterar;
      tamanho = 16;
      lcd.setCursor(0, 0);
    }
```

```
    lcd.print(nome.substring(inicio, inicio + tamanho));
    inicio++;
  }
  if (inicio > nome.length()) {
    inicio = 0;
    tamanho = 1;
    alterar = !alterar;
  }
  delay(TEMPO_ATUALIZACAO);
}
```

Neste programa, como podemos observar no código-fonte anterior, vamos aplicar os conceitos sobre display de LCD, abordados no sketch anterior, para "rolar" uma mensagem de texto no display de LCD.

7.2 Displays de LED com sete segmentos



PROJETO Nº 14 Utilização do display de LED

O objetivo deste projeto é demonstrar a utilização do display de LED de sete segmentos, controlado diretamente a partir das portas do Arduino. Eles são bastante comuns e muitos utilizados para exibir, principalmente, informações numéricas. Conforme pode ser observado na Figura 7.3, esses displays podem ser do tipo cátodo comum ou ânodo comum.

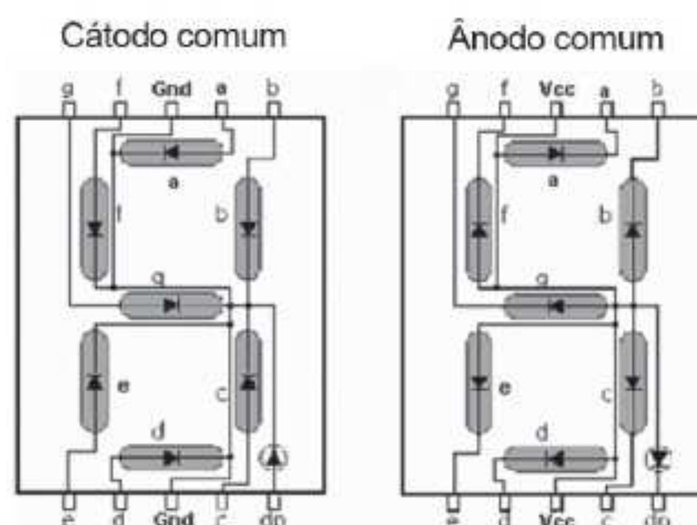


Figura 7.3 – Displays de LEDs com sete segmentos.



Material necessário

- 1 Arduino;
- 1 display de LED de sete segmentos (um dígito);
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

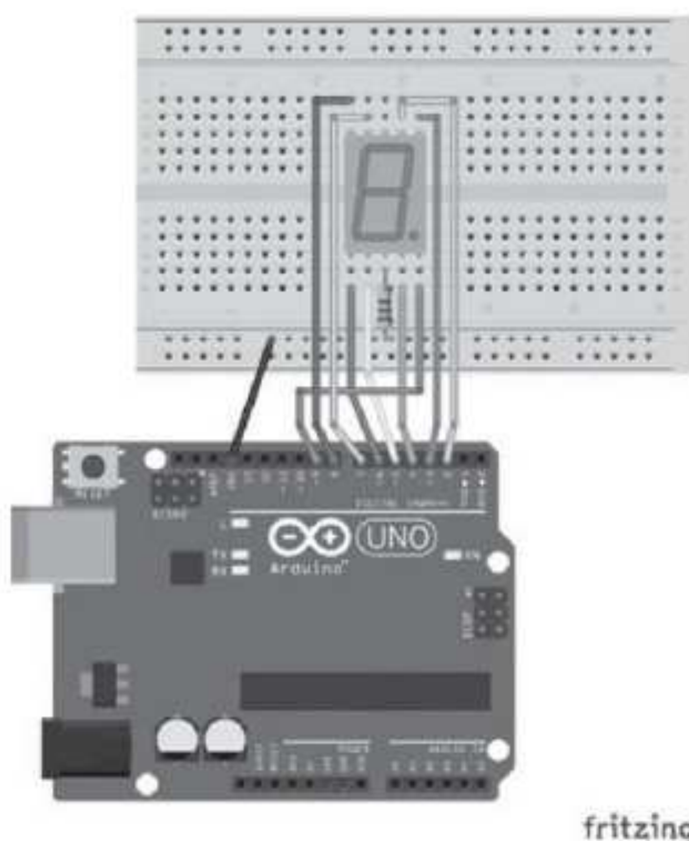


Figura 7.4 – Conexão do display de LED de sete segmentos ao Arduino.

Adotando como referência a Figura 7.4, realize a sequência de montagem:

a) Pino 1 (segmento e) do display ligado ao 6 do Arduino.
b) Pino 2 (segmento d) do display ligado ao 5 do Arduino.
c) Pino 3 (GND, se cátodo comum ou VCC se ânodo comum) do display ligado ao resistor de 220 ohms (ou 330 ohms).
d) Resistor de 220 ohms (ou 330 ohms) ligado ao GND (se cátodo comum) ou VCC (se ânodo comum) do Arduino.
e) Pino 4 (segmento c) do display ligado ao 4 do Arduino.
f) Pino 5 (ponto decimal) do display ligado ao 9 do Arduino.

g) Pino 6 (segmento b) do display ligado ao 3 do Arduino.
h) Pino 7 (segmento a) do display ligado ao 2 do Arduino.
i) Pino 9 (segmento f) do display ligado ao 7 do Arduino.
j) Pino 10 (segmento g) do display ligado ao 8 do Arduino.



Programa

No ambiente de desenvolvimento do Arduino, implemente o sketch a seguir:

```
int SEG_A = 2;
int SEG_B = 3;
int SEG_C = 4;
int SEG_D = 5;
int SEG_E = 6;
int SEG_F = 7;
int SEG_G = 8;
int PONTO = 9;

int ATRASO = 1000;

void setup() {
  pinMode(SEG_A, OUTPUT);
  pinMode(SEG_B, OUTPUT);
  pinMode(SEG_C, OUTPUT);
  pinMode(SEG_D, OUTPUT);
  pinMode(SEG_E, OUTPUT);
  pinMode(SEG_F, OUTPUT);
  pinMode(SEG_G, OUTPUT);
  pinMode(PONTO, OUTPUT);
}

void loop() {
  digitalWrite(SEG_B, HIGH);
  digitalWrite(SEG_C, HIGH);
  delay(ATRASO);
}
```

Nesse programa, definimos cada um dos pinos do Arduino que está conectado a um determinado segmento do display de LED. Na função setup, definimos esses pinos como saída (OUTPUT). No trecho de programa a seguir, podemos observar que dentro da função loop vamos acender os segmentos desejados. Dessa forma, para exibirmos no display o número um, colocamos em nível um (HIGH) nos segmentos B e C.

```
digitalWrite(SEG_B, HIGH);
digitalWrite(SEG_C, HIGH);
```

Então, seguindo o mesmo raciocínio, para apresentar o dígito quatro devemos colocar em nível HIGH os segmentos B, C, F e G.

```
digitalWrite(SEG_B, HIGH);
digitalWrite(SEG_C, HIGH);
digitalWrite(SEG_F, HIGH);
digitalWrite(SEG_G, HIGH);
```



Programa

Mantenha a mesma montagem utilizada no programa anterior e, no ambiente de desenvolvimento do Arduino, implemente o sketch a seguir:

```
int SEG_A = 2;
int SEG_B = 3;
int SEG_C = 4;
int SEG_D = 5;
int SEG_E = 6;
int SEG_F = 7;
int SEG_G = 8;
int PONTO = 9;

int ATRASO = 150;

void setup() {
  for (int pino = SEG_A; pino <= SEG_G; pino++) {
    pinMode(pino, OUTPUT);
  }
}

void loop() {
  for (int pino = SEG_A; pino < SEG_G; pino++) {
    digitalWrite(pino, HIGH);
    if (pino > SEG_A)
      digitalWrite(pino - 1, LOW);
    else
      digitalWrite(SEG_F, LOW);
    delay(ATRASO);
  }
}
```

Neste exemplo, vamos criar um pequeno efeito de animação acendendo e apagando os segmentos em ordem sequencial, ou seja, acendemos o segmento A por 150 ms, em seguida apagamos o A e acendemos o B; após 150 ms apagamos o B e acendemos o C, e assim sucessivamente. Observe também que neste exemplo o segmento G não é utilizado.



Programa

Este próximo exemplo mostrará uma contagem regressiva de nove até zero no display de LED. Também utilizando o mesmo circuito dos exemplos anteriores, vá até o ambiente de desenvolvimento do Arduino e digite o seguinte sketch:

```
// Declaração da matriz para uso em displays com cátodo comum, para utilizar em
displays com ânodo comum, basta inverter o valor dos bytes.

byte digitos[10][7] = {
  { 1,1,1,1,1,1,0 },  // = 0
  { 0,1,1,0,0,0,0 },  // = 1
  { 1,1,0,1,1,0,1 },  // = 2
  { 1,1,1,1,0,0,1 },  // = 3
  { 0,1,1,0,0,1,1 },  // = 4
  { 1,0,1,1,0,1,1 },  // = 5
  { 1,0,1,1,1,1,1 },  // = 6
  { 1,1,1,0,0,0,0 },  // = 7
  { 1,1,1,1,1,1,1 },  // = 8
  { 1,1,1,0,0,1,1 }   // = 9
};

void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pontoDecimal(false);
}
```

```
void pontoDecimal(boolean ponto) {
  digitalWrite(9, ponto);
}

void escrever(int digito) {
  int pino = 2;
  for (int segmento = 0; segmento < 7; segmento++) {
    digitalWrite(pino, digitos[digito][segmento]);
    pino++;
  }
}

void limpar() {
  byte pino = 2;
  for (int segmento = 0; segmento < 7; segmento++) {
    digitalWrite(pino, LOW);
    pino++;
  }
}

void loop() {
  for (int cont = 9; cont >= 0; cont--) {
    escrever(cont);
    delay(1000);
  }
  limpar();
  delay(1000);
}
```

Neste exemplo, vamos exibir os números entre zero e nove no display de LEDs. Para isso, inicialmente devemos definir uma matriz de bytes que indicaram o estado (aceso ou apagado) de cada um dos sete segmentos do display. Dessa forma, como podemos observar no trecho de programa a seguir, temos a definição dos dez dígitos que serão utilizados.

```
byte digitos[10][7] = {
  { 1,1,1,1,1,1,0 },  // = 0
  { 0,1,1,0,0,0,0 },  // = 1
  { 1,1,0,1,1,0,1 },  // = 2
  { 1,1,1,1,0,0,1 },  // = 3
  { 0,1,1,0,0,1,1 },  // = 4
  { 1,0,1,1,0,1,1 },  // = 5
```

```
  { 1,0,1,1,1,1,1 },  // = 6
  { 1,1,1,0,0,0,0 },  // = 7
  { 1,1,1,1,1,1,1 },  // = 8
  { 1,1,1,0,0,1,1 }   // = 9
};
```

A função escrever, detalhada a seguir, receberá como parâmetro um dígito e a partir daí utiliza o laço de repetição for para definir o valor (0 ou 1) para cada um dos segmentos.

```
void escrever(int digito) {
  int pino = 2;
  for (int segmento = 0; segmento < 7; segmento++) {
    digitalWrite(pino, digitos[digito][segmento]);
    pino++;
  }
}
```

Conforme podemos observar a seguir, a função limpar apagará todos os segmentos do display:

```
void limpar() {
  byte pino = 2;
  for (int segmento = 0; segmento < 7; segmento++) {
    digitalWrite(pino, LOW);
    pino++;
  }
}
```

Na função loop mostrada a seguir, realizamos a contagem regressiva exibindo cada dígito por 1.000 ms, ou seja, 1 segundo. Ao final da contagem, o display é apagado por 1 segundo, e, após este intervalo de tempo, a contagem regressiva se iniciará novamente.

```
void loop() {
  for (int cont = 9; cont >= 0; cont--) {
    escrever(cont);
    delay(1000);
  }
  limpar();
  delay(1000);
}
```

7.3 O circuito integrado MAX 7219 ou 7221



PROJETO Nº 15
Uso de displays com múltiplos dígitos

Você já deve ter observado pelos exemplos anteriores que, quando precisamos utilizar displays de LEDs, os segmentos acabam por consumir muitas das portas do Arduino. Dessa forma, quando é necessário utilizar displays que possuem mais do que um dígito, as portas disponíveis no Arduino não serão suficientes, ou, mesmo que sejam, não permitirão colocar novas funcionalidades ao seu projeto, como, por exemplo, um sensor de temperatura ou um módulo de relógio em tempo real (RTC). Assim, para otimizar o uso das portas do Arduino, devemos utilizar um driver para displays de LED. O mais popular é o Maxim 7219 ou 7221, cuja pinagem pode ser observada na Figura 7.5.

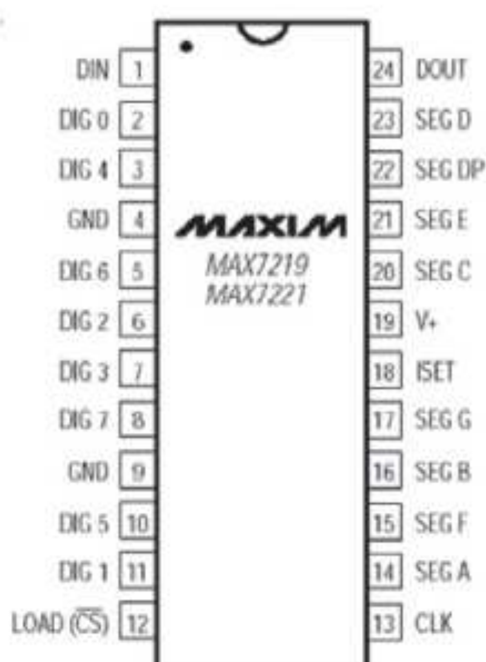


Figura 7.5 – Pinagem do MAX 7219 ou 7221.



Material necessário

- 1 Arduino;
- 1 display de LED de sete segmentos (quatro dígitos);
- 1 circuito integrado (CI) MAX 7219 ou 7221;
- 1 resistor de 100 kohms (marrom, preto, amarelo);
- 1 protoboard;
- jumper cable.



Montagem do circuito

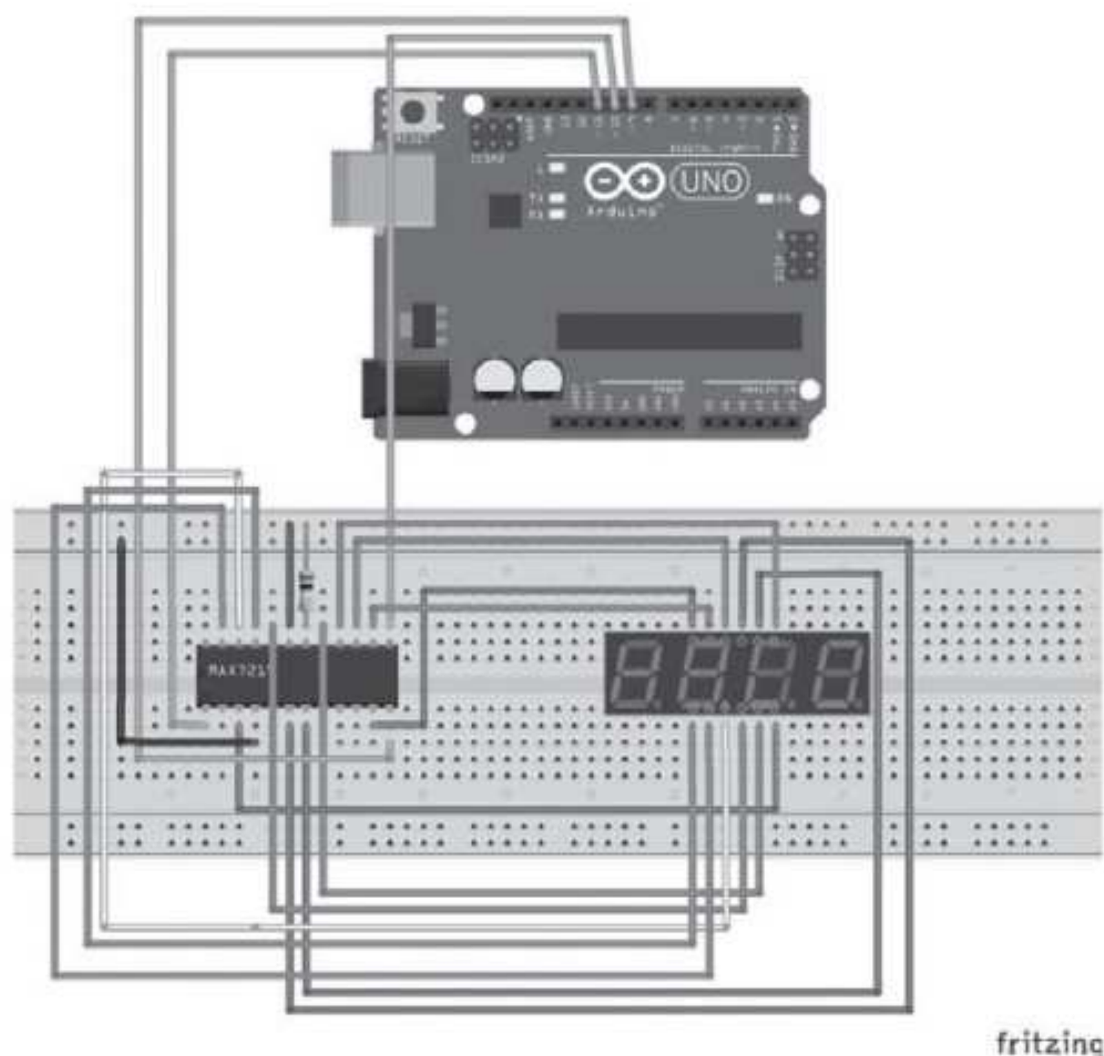


Figura 7.6 – Uso do CI MAX 7219 ou 7221.

Conforme ilustrado no diagrama mostrado na Figura 7.6, realize os próximos passos necessários para a montagem do projeto:

a) Conecte o pino 1 (DIN) do MAX 7219/21 ao pino 11 do Arduino.
b) Conecte o pino 3 (DIG 4) do MAX 7219/21 ao pino 6 do display.
c) Conecte o pino 4 (GND) do MAX 7219/21 à linha de alimentação negativa da protoboard.
d) Conecte o pino 6 (DIG 2) do MAX 7219/21 ao pino 9 do display.
e) Conecte o pino 7 (DIG 3) do MAX 7219/21 ao pino 8 do display.
f) Conecte o pino 11 (DIG 1) do MAX 7219/21 ao pino 12 do display.
g) Conecte o pino 12 (LOAD/CS) do MAX 7219/21 ao pino 9 do Arduino.
h) Conecte o pino 13 (CLK) do MAX 7219/21 ao pino 10 do Arduino.
i) Conecte o pino 14 (SEG A) do MAX 7219/21 ao pino 11 do display.
j) Conecte o pino 15 (SEG F) do MAX 7219/21 ao pino 10 do display.
k) Conecte o pino 16 (SEG B) do MAX 7219/21 ao pino 7 do display.

l) Conecte o pino 17 (SEG G) do MAX 7219/21 ao pino 5 do display.

m) Conecte o pino 18 (SEG F) do MAX 7219/21 ao resistor de 100 kohms.

n) Conecte o outro terminal do resistor à linha de alimentação positiva (5 V) da protoboard.

o) Conecte o pino 19 (V+) do MAX 7219/21 à linha de alimentação positiva (5 V) da protoboard.

p) Conecte o pino 20 (SEG C) do MAX 7219/21 ao pino 4 do display.

q) Conecte o pino 21 (SEG E) do MAX 7219/21 ao pino 1 do display.

r) Conecte o pino 22 (SEG DP) do MAX 7219/21 ao pino 13 da matriz.

s) Conecte o pino 23 (SEG D) do MAX 7219/21 ao pino 6 da matriz.

t) Conecte o pino 5V do Arduino à linha de alimentação positiva da protoboard.

u) Conecte o pino GND do Arduino à linha de alimentação negativa da protoboard.



Programa

Este sketch exibe o número 1234 em um display de LED de sete segmentos com quatro dígitos utilizando, para isso, a biblioteca LEDControl.

```
#include "LEDControl.h"

/*
 * Criar um LEDControl (lc).
 * O pino 11 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX7219/21
 * O pino 10 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX7219/21
 * O pino 9 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX7219/21
 * O quarto parâmetro indica que há apenas um MAX7219/21
 * conectado ao Arduino
*/
LEDControl lc = LEDControl(11, 10, 9, 1);

void setup() {
  // Retira o MAX7219/21 do modo de economia de energia
  lc.shutdown(0, false);

  // Define a intensidade do brilho dos LEDs
  lc.setIntensity(0, 2);
  lc.clearDisplay(0);
}
```

```
void loop() {
lc.setChar(0, 2, '0', false);
  lc.setChar(0, 2, '1', false);
  lc.setChar(0, 3, '2', false);
  lc.setChar(0, 4, '3', false);
}
```

Conforme mostrado no trecho de programa a seguir, inicialmente vamos criar um objeto a partir da classe LEDControl. Para realizar isso, devemos informar os pinos do Arduino conectados ao DATA IN, CLK e LOAD CS do circuito integrado MAX 7219 ou 7221. Também é necessário informar a quantidade de circuitos integrados MAX 7219 ou 7221 utilizados no circuito, que neste exemplo é apenas um.

```
LEDControl lc = LEDControl(11, 10, 9, 1);
```

Na função setup retiramos o MAX 7219 ou 7221 do modo de economia de energia, definimos a intensidade luminosa dos LEDs do display e, por último, utilizamos o método clearDisplay para apagar todos os segmentos dele. Observe que o primeiro parâmetro dos métodos shutdown, setIntensity e clearDisplay se refere ao número do MAX 7219 ou 7221 que será afetado pelo comando. No nosso projeto, como usamos um único circuito integrado, este valor é sempre zero, que se refere ao primeiro circuito integrado. Se tivéssemos um segundo circuito integrado conectado, esse seria o número um, e assim sucessivamente.

```
void setup() {
  // Retira o MAX7219/21 do modo de economia de energia
  lc.shutdown(0, false);

  // Define a intensidade do brilho dos LEDs
  lc.setIntensity(0, 2);
  lc.clearDisplay(0);
}
```



Programa

Neste novo sketch será realizada a exibição dos números inteiros entre -999 e 999 em um display de LED de sete segmentos com quatro dígitos.

```
#include "LEDControl.h"

/*
 * Criar um LEDControl (lc).
 * O pino 11 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 10 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 9 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino
*/
LEDControl lc = LEDControl(11, 10, 9, 1);

int i = -999;

void setup() {
  // Retira o MAX 7219/21 do modo de economia de energia
  lc.shutdown(0, false);

  // Define a intensidade do brilho dos LEDs
  lc.setIntensity(0, 2);

  lc.clearDisplay(0);
}

void loop() {
  exibirInteiro(i++);
}

void exibirInteiro(int valor) {
  int unidade;
  int dezena;
  int centena;
  boolean negativo = false;

  if(valor < -999 || valor > 999)
    return;

  if(valor < 0) {
    negativo = true;
    valor = valor * (-1);
  }
```

```
  unidade = valor % 10;
  valor = valor / 10;
  dezena = valor % 10;
  valor = valor / 10;
  centena = valor;

  if (negativo) {
    // Exibe o sinal de negativo no display que está
    //mais à esquerda
    lc.setChar(0, 1, '-', false);
  }
  else {
    // Exibe um espaço no lugar do sinal de negativo
    lc.setChar(0, 1 , ' ', false);
  }

  // Exibe cada um dos dígitos que compõem o número
  lc.setDigit(0, 2, (byte)centena, false);
  lc.setDigit(0, 3, (byte)dezena, false);
  lc.setDigit(0, 4, (byte)unidade, false);
  delay(100);
}
```

O conceito básico por trás deste exemplo é realizar a decomposição de um determinado valor numérico em centenas, dezenas e unidades, conforme ilustrado pelo trecho de programa a seguir:

```
unidade = valor % 10;
valor = valor / 10;
dezena = valor % 10;
valor = valor / 10;
centena = valor;
```

Em seguida, exibimos em cada um dos dígitos o respectivo componente pela utilização do método setDigit, ou seja:

```
// Exibe cada um dos dígitos que compõem o número
lc.setDigit(0, 2, (byte)centena, false);
lc.setDigit(0, 3, (byte)dezena, false);
lc.setDigit(0, 4, (byte)unidade, false);
```

Observe que o método setDigit recebe como parâmetros o número do circuito integrado, o número do dígito, o valor a ser exibido e se o ponto decimal deverá ou não ser

ligado. Também é importante salientar que, neste exemplo, o display número 1 será empregado para mostrar o sinal negativo apenas. A seguir, temos o trecho do programa que realizará este trabalho:

```
if (negativo) {
  // Exibe o sinal de negativo no display que está
  // mais à esquerda
  lc.setChar(0, 1, '-', false);
}
```

7.4 Matriz de LEDs

As matrizes de LEDs são bastante populares em letreiros e consistem em um conjunto de LEDs encapsulados formando um único componente. Os LEDs podem ser endereçados individualmente por meio da respectiva linha e coluna. O circuito integrado MAX 7219 ou 7221 e a biblioteca LEDControl também podem ser utilizados para facilitar o uso de matrizes de LEDs.



PROJETO Nº 16
Utilização da matriz de LEDs

Neste projeto vamos abordar os conceitos básicos sobre matrizes de LEDs, ou seja, o endereçamento de um LED da matriz por meio de linha e coluna e também os conceitos necessários à exibição de caracteres (letras, números ou qualquer outro símbolo) na matriz.



Material necessário

- 1 Arduino;
- 1 matriz de LED de 8x8*;
- 1 CI MAX 7219 ou 7221*;
- 1 resistor de 100 kohms (marrom, preto, amarelo)*;
- 1 protoboard;
- jumper cable.

* Podem ser substituídos por um módulo de matriz de LEDs.



Montagem do circuito

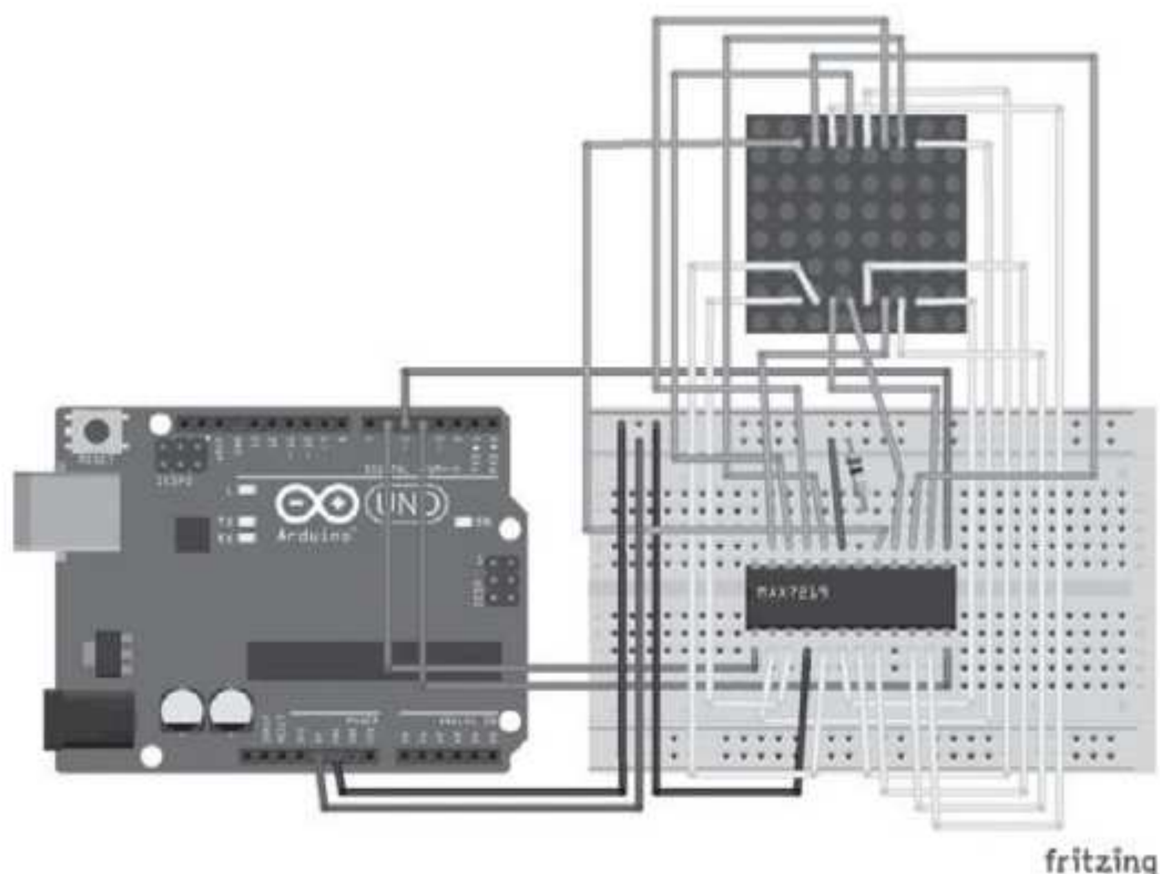


Figura 7.7 – Projeto com matriz de LEDs.

Conforme ilustrado no diagrama mostrado na Figura 7.7, realize os passos necessários para a montagem do projeto:

a) Conecte o pino 1 (DIN) do MAX 7219/21 ao pino 6 do Arduino.
b) Conecte o pino 2 (DIG 0) do MAX 7219/21 ao pino 9 da matriz.
c) Conecte o pino 3 (DIG 4) do MAX 7219/21 ao pino 1 da matriz.
d) Conecte o pino 4 (GND) do MAX 7219/21 à linha de alimentação negativa da protoboard.
e) Conecte o pino 5 (DIG 6) do MAX 7219/21 ao pino 2 da matriz.
f) Conecte o pino 6 (DIG 2) do MAX 7219/21 ao pino 8 da matriz.
g) Conecte o pino 7 (DIG 3) do MAX 7219/21 ao pino 12 da matriz.
h) Conecte o pino 8 (DIG 7) do MAX 7219/21 ao pino 5 da matriz.
i) Conecte o pino 10 (DIG 5) do MAX 7219/21 ao pino 7 da matriz.
j) Conecte o pino 11 (DIG 1) do MAX 7219/21 ao pino 14 da matriz.
k) Conecte o pino 12 (LOAD/CS) do MAX 7219/21 ao pino 4 do Arduino.
l) Conecte o pino 13 (CLK) do MAX 7219/21 ao pino 5 do Arduino.
m) Conecte o pino 14 (SEG A) do MAX 7219/21 ao pino 3 da matriz.
n) Conecte o pino 15 (SEG F) do MAX 7219/21 ao pino 15 da matriz.

o) Conecte o pino 16 (SEG B) do MAX 7219/21 ao pino 4 da matriz.
p) Conecte o pino 17 (SEG G) do MAX 7219/21 ao pino 16 da matriz.
q) Conecte o pino 18 (SEG F) do MAX 7219/21 ao resistor de 100 kohms.
r) Conecte o outro terminal do resistor à linha de alimentação positiva (5 V) da protoboard.
s) Conecte o pino 19 (V+) do MAX 7219/21 à linha de alimentação positiva (5 V) da protoboard.
t) Conecte o pino 20 (SEG C) do MAX 7219/21 ao pino 10 da matriz.
u) Conecte o pino 21 (SEG E) do MAX 7219/21 ao pino 11 da matriz.
v) Conecte o pino 22 (SEG DP) do MAX 7219/21 ao pino 13 da matriz.
w) Conecte o pino 23 (SEG D) do MAX 7219/21 ao pino 6 da matriz.
x) Conecte o pino 5 V do Arduino à linha de alimentação positiva da protoboard.
y) Conecte o pino GND do Arduino à linha de alimentação negativa da protoboard.

Com o intuito de facilitar esse tipo de montagem, existem alguns módulos prontos que já apresentam a conexão do MAX 7219/21 com a matriz de LEDs. Dessa forma, conforme ilustrado pela Figura 7.8, apenas nos preocupamos com a conexão do módulo com o Arduino, visto que as conexões entre o CI e a matriz de LEDs já estão estabelecidas na placa de circuito impresso do módulo.

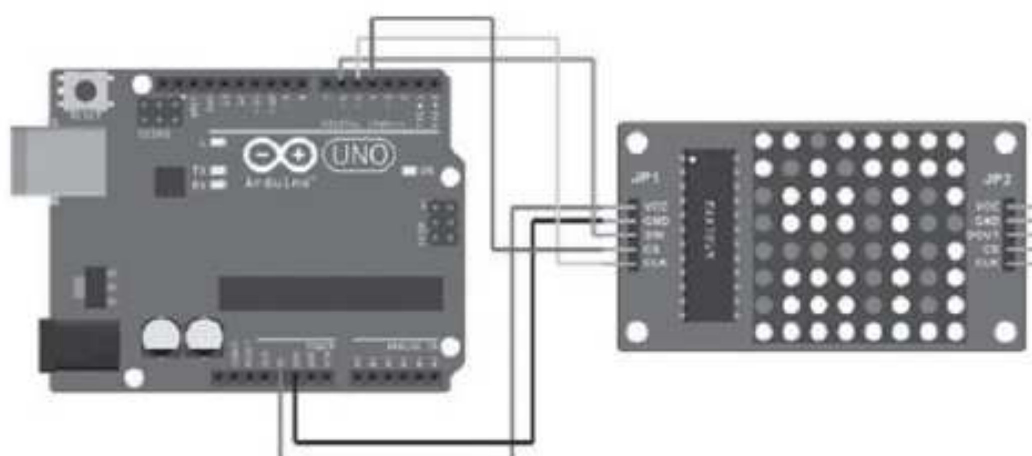


Figura 7.8 – Uso de um módulo de matriz de LEDs.

Caso você faça a opção pela montagem utilizando o módulo (Figura 7.8), realize os seguintes passos para a construção do projeto:

a) Conecte o pino VCC do módulo ao pino 5 V do Arduino.
b) Conecte o pino GND do módulo ao pino GND do Arduino.
c) Conecte o pino DIN (Data In) do módulo ao pino 6 do Arduino.
d) Conecte o pino CS (LOAD/CS) do módulo ao pino 4 do Arduino.
e) Conecte o pino CLK (CLOCK) do módulo ao pino 5 do Arduino.

Também é importante salientar que os programas que serão desenvolvidos a seguir se aplicam aos dois tipos de montagem, pois referem-se ao mesmo circuito eletrônico.



Programa

Este sketch criará um efeito de acendimento dos LEDs parecido com o existente no visor dos Cylons de Battlestar Galactica. Um LED será aceso na primeira coluna de uma determinada linha e irá "mover-se" para a coluna seguinte; ao atingir a última, realizará o "movimento" inverso. Quando atingir a primeira coluna, novamente o ciclo se repete.

```
#include "LEDControl.h"

#define ATRASO 100

/*
 * Criar um LEDControl (matrizLED).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino
*/
LEDControl matrizLED = LEDControl(6, 5, 4, 1);

void setup() {
// Retira o MAX 7219/21 do modo de economia de energia
  matrizLED.shutdown(0, false);

  // Define a intensidade do brilho dos LEDs
matrizLED.setIntensity(0, 2);

  // Apagar o conteúdo exibido na matriz de LEDs
matrizLED.clearDisplay(0);
}

void loop() {
 for (int i = 0; i < 8; i++) {
// Acende o LED que está na linha e coluna
matrizLED.setLED(0, i, 4, HIGH);
```

```
    delay(ATRASO);

// Apaga o LED que está na linha e coluna
matrizLED.setLED(0, i, 4, LOW);
}

  for (int i = 7; i >= 0; i--) {
 matrizLED.setLED(0, i, 4, HIGH);
delay(ATRASO);
    matrizLED.setLED(0, i, 4, LOW);
  }

  matrizLED.clearDisplay(0);
}
```

Observando este programa, podemos observar que a utilização da biblioteca LEDControl para controlar uma matriz de LEDs é bastante similar aos exemplos anteriores, nos quais utilizamos a referida biblioteca para o controle de displays de LEDs de sete segmentos. Para definir qual LED deve ser aceso ou apagado, utilizamos o método setLED no qual passamos, como parâmetros, o número do display, a linha, a coluna e se o LED deverá ser aceso (HIGH) ou desligado (LOW). A primeira estrutura de repetição for fará o efeito de movimento da esquerda para a direita, enquanto o segundo for realizará o movimento contrário, ou seja, da direita à esquerda.



Programa

Neste novo sketch vamos "escrever" na matriz de LEDs uma letra. Para isso, torna-se necessário carregarmos um vetor contendo cada um dos caracteres que podem ser exibidos. Por exemplo, para obtermos a letra "A", devemos especificar os seguintes valores para a matriz:

```
int letraA[] = {
  B01110000,
  B10001000,
  B10001000,
  B10001000,
  B11111000,
  B10001000,
  B10001000 };
```

Observe que o B na frente do número indica que estamos especificando-o em valores binários, ou seja, zero ou um. Note também que os bits que estão com o valor um (HIGH) indicam os LEDs que estarão acesos e formarão o caractere. Para facilitar o entendimento, vamos deixar apenas os bits um do vetor acima, trocando, para fins didáticos, apenas os demais símbolos por espaço em branco. Dessa forma, como podemos observar a seguir, fica mais fácil visualizar o caractere "A" que será exibido.

```
 111
1   1
1   1
1   1
11111
1   1
1   1
```

Digite o sketch a seguir no ambiente de desenvolvimento para observar a exibição do caractere "A" na matriz de LEDs.

```
#include "LEDControl.h"

int letraA[] = {
  B01110000,
  B10001000,
  B10001000,
  B10001000,
  B11111000,
  B10001000,
  B10001000 };

/*
 * Criar um LEDControl (matrizLED).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino
 */
LEDControl lc=LEDControl(6, 5, 4, 1);
```

```
void setup() {
  lc.shutdown(0, false);
  lc.setIntensity(0, 2);
  lc.clearDisplay(0);
}

void loop() {
  for (int i = 0; i < 7; i++) {
    lc.setColumn(0, 7 - i, letraA[i]);
  }
}
```

Após criarmos um objeto instanciado a partir da classe LEDControl e iniciarmos o circuito integrado MAX 7219/21 na função setup, utilizamos loop para exibir cada uma das linhas do vetor que irão formar o caractere a ser exibido na matriz de LEDs. O método setColumn será responsável pela exibição dos dados da linha na matriz de LEDs.



Programa

No programa a seguir vamos "escrever" na matriz de LEDs cada uma das letras de uma cadeia de caracteres (String). Porém, para relizarmos isso, devemos carregar, em um vetor, não apenas uma única letra, mas todos os caracteres imprimíveis. Como é possível imaginar, esse vetor será bastante extenso e poderá ocupar praticamente toda a memória volátil (RAM) do Arduino, inviabilizando a sua utilização em sketches mais complexos e extensos. Para evitarmos isso, o Arduino nos oferece a possibilidade de armazenar dados na sua memória flash, que possui ...Mbytes, em vez de mantê-los na memória RAM, por meio da utilização da biblioteca avr/pgmspace.h.

Nesse caso, quando declaramos a variável ou o vetor que será mantido na memória flash, devemos utilizar a palavra reservada PROGMEN após a declaração da variável ou vetor, por exemplo:

```
prog_uchar fonte5x7 [] PROGMEM = {
    B00000000,       //Space (Char 0x20)
    B00000000,
    B00000000,
    B00000000,
    B00000000,
    B00000000,
    B00000000,

    // Definir os demais caracteres imprimíveis ...
};
```

Posteriormente, para obtermos o dado armazenado na memória flash devemos usar a função pgm_read_byte_near, ou seja:

```
int dado = pgm_read_byte_near(fonte5x7[0]);
```

É uma boa prática colocar tal extenso vetor em uma biblioteca, podendo ser reutilizado em outros sketches que precisarão exibir texto em uma matriz de LEDs. Dessa forma, no presente exemplo, o vetor estará no arquivo FonteMatriz.h e deverá ser carregado em nosso projeto pela diretiva include:

```
#include "FonteMatriz.h"
```

Após esses conceitos iniciais, digite no ambiente de desenvolvimento do Arduino o programa a seguir:

```
#include <avr/pgmspace.h>
#include "LEDControl.h"
#include "FonteMatriz.h"

const int NUM_MATRIZ = 1;
const int ATRASO = 500;

/*
 * Criar um LEDControl (matrizLED).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino (NUM_MATRIZ)
 */
LEDControl lc=LEDControl(6, 5, 4, NUM_MATRIZ);

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < NUM_MATRIZ; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 2);
    lc.clearDisplay(i);
  }
}
```

```
void loop() {
  String texto = "Arduino";
  exibirMensagem(texto);
}

void exibirMensagem(String mensagem) {
  int contador = 0;
  int caractere = 0;

  do {
    // Obtém cada caractere que compõe a mensagem
    caractere =  mensagem.charAt(contador);
    if (caractere != 0) {
      exibirCaractere(caractere);
      delay(ATRASO);
    }
    contador++;
  }
  while (caractere != 0);
}

void exibirCaractere(int ascii) {
  if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
    for (int i = 0; i < 7; i++) {
      // Busca na tabela de caracteres uma linha
      intlinha =
        pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
      lc.setColumn(0, 7 - i, linha);
    }
  }
}
```

Conforme podemos observar no próximo trecho de programa, após inicializarmos o circuito integrado MAX 7219 ou 7221 devemos, na função loop, obter separadamente cada caractere que será exibido. Conseguimos fazer isso utilizando o método charAt, disponível para objetos instanciados a partir da classe String. Em seguida, cada caractere é enviado para a função exibirCaractere.

```
do {
    // Obtém cada caractere que compõe a mensagem
    caractere =  mensagem.charAt(contador);
    if (caractere != 0) {
      exibirCaractere(caractere);
      delay(ATRASO);
    }
    contador++;
  }
  while (caractere != 0);
```

A função exibirCaractere, mostrada a seguir, irá verificar se o caractere é imprimível, ou seja, maior ou igual a 0x20 (espaço em branco) e menor ou igual a 0x7f (° - símbolo de graus). Em seguida, por meio da função pgm_read_byte_near, recuperará cada uma das linhas que comporão o caractere a ser exibido.

```
void exibirCaractere(int ascii) {
  if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
    for (int i = 0; i < 7; i++) {
      // Busca na tabela de caracteres uma linha
      int linha =
        pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
      lc.setColumn(0, 7 - i, linha);
    }
  }
}
```

Concluindo a análise deste código, utilizaremos o método setColumn, dentro do laço de repetição for, para exibir os dados da linha na matriz de LEDs, sendo uma linha por vez, até completar o caractere.



PROJETO Nº 17
Rolagem do texto na matriz de LEDs

Ampliando os conceitos sobre matriz de LEDs abordados até agora, vamos realizar uma montagem que utilizará duas matrizes de LEDs e faremos o texto "rolar" por elas da direita para a esquerda. A biblioteca LEDControl permite que até oito módulos sejam conectados.



Material necessário

- 1 Arduino;
- 2 matrizes de LED de 8x8*;
- 2 CI MAX 7219 ou 7221*;
- 2 resistores de 100 kohms (marrom, preto, amarelo)*;
- 1 protoboard;
- jumper cable.

* Podem ser substituídos por dois módulos de matriz de LEDs.



Montagem do circuito

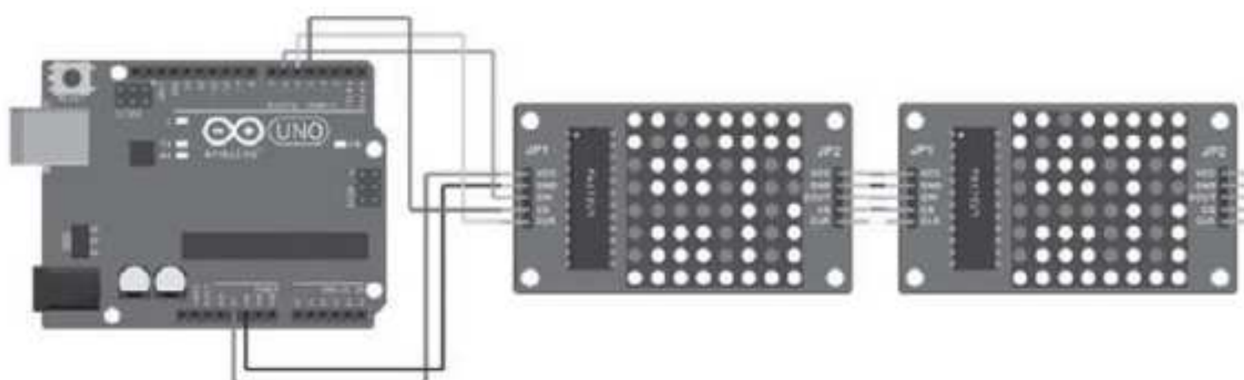


Figura 7.9 – Projeto de rolagem de texto em matrizes de LEDs.

Conforme ilustrado no diagrama mostrado na Figura 7.9, realize os passos necessários para a montagem do projeto:

a) Conecte o pino VCC do módulo ao pino 5 V do Arduino.
b) Conecte o pino GND do módulo ao pino GND do Arduino.
c) Conecte o pino DIN (Data In) do primeiro módulo ao pino 6 do Arduino.
d) Conecte o pino CS (LOAD/CS) do primeiro módulo ao pino 4 do Arduino.
e) Conecte o pino CLK (CLOCK) do primeiro módulo ao pino 5 do Arduino.
f) Conecte o pino DOUT (Data Out) do primeiro módulo ao pino DIN (Data In) do segundo módulo.
g) Conecte o pino CS (LOAD/CS) do primeiro módulo ao pino CS do segundo módulo.
h) Conecte o pino CLK (CLOCK) do primeiro módulo ao pino CLK do segundo módulo.



Programa

Digite o sketch a seguir no ambiente de desenvolvimento do Arduino, lembrando de inserir no projeto as bibliotecas utilizadas no exemplo anterior, ou seja, avr/pgmspace.h, LEDControl.h e FonteMatriz.h.

```
#include <avr/pgmspace.h>
#include "LEDControl.h"
#include "FonteMatriz.h"

const int NUM_MATRIZ = 2;
const int ATRASO = 500;

/*
 * Criar um LEDControl (matrizLED).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino (NUM_MATRIZ)
 */
LEDControl lc=LEDControl(6, 5, 4, NUM_MATRIZ);

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < NUM_MATRIZ; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 2);
    lc.clearDisplay(i);
  }
}

void loop() {
  String texto = "Arduino";
  rolarMensagem(texto);
}
```

```
void rolarMensagem(String mensagem) {
  int contador = 0;
  int caractere = 0;
  int caractereAnterior = 0;

  do {
    // Obtém cada caractere que compõe a mensagem
    caractere =  mensagem.charAt(contador);
    if (caractere != 0) {
      exibirCaractere(1, caractere);
      exibirCaractere(0, caractereAnterior);
      caractereAnterior = caractere;
      delay(ATRASO);
    }
    contador++;
  }
  while (caractere != 0);
  lc.clearDisplay(0);
}

void exibirCaractere(int matriz, int ascii) {
if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
    for (int i = 0; i < 7; i++) {
      // Busca na tabela de caracteres uma linha
      intlinha =
        pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
      lc.setColumn(matriz, 7 - i, linha);
    }
  }
}
```

Conforme podemos ver no trecho de código-fonte a seguir e como o presente projeto trabalha com duas matrizes de LEDs, precisamos inicializar as duas matrizes e os respectivos MAX 7219 ou 7221 que serão utilizados.

```
const int NUM_MATRIZ = 2;

LEDControl lc=LEDControl(6, 5, 4, NUM_MATRIZ);

void setup() {
  Serial.begin(9600);
```

```
  for (int i = 0; i < NUM_MATRIZ; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 2);
    lc.clearDisplay(i);
  }
}
```

Analisando o código da função rolarMensagem, mostrada a seguir, podemos observar que ela recebe uma String (cadeia de caracteres) como parâmetro e irá exibir o caractere atual na matriz número 1 e aquele obtido anteriormente na matriz 0, criando um efeito de rolagem dos caracteres da direita à esquerda.

```
void rolarMensagem(String mensagem) {
int contador = 0;
  int caractere = 0;
  int caractereAnterior = 0;

  do {
    // Obtém cada caractere que compõe a mensagem
    caractere =  mensagem.charAt(contador);
    if (caractere != 0) {
      exibirCaractere(1, caractere);
      exibirCaractere(0, caractereAnterior);
      caractereAnterior = caractere;
      delay(ATRASO);
    }
    contador++;
  }
  while (caractere != 0);
  lc.clearDisplay(0);
}
```

Note que a função exibirCaractere receberá como parâmetros a matriz que exibirá o caractere que será obtido da memória flash e também aquele a ser mostrado, conforme podemos ver no trecho de programa a seguir.

```
void exibirCaractere(int matriz, int ascii) {
  if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
```

```
      for (int i = 0; i < 7; i++) {
        // Busca na tabela de caracteres uma linha
        int linha =
          pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
        lc.setColumn(matriz, 7 - i, linha);
      }
    }
  }
```

Exercícios

1. A partir do circuito mostrado na Figura 7.10, elabore uma rotina que mostre o seu nome completo.

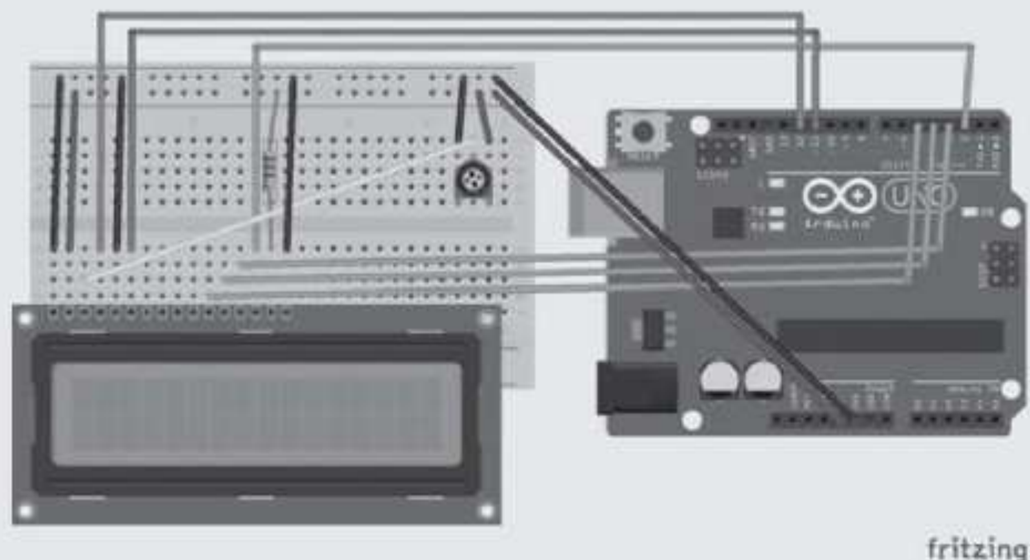


Figura 7.10 – Circuito básico para uso do LCD.

2. Utilizando como referência o circuito mostrado na Figura 7.10, escreva um sketch que mostre todas as letras de alfabeto em letras maiúsculas e, em seguida, em minúsculas.
3. Em uma matriz de LEDs (Figura 7.11), mostre as letras do seu primeiro nome com intervalo de um segundo entre uma letra e outra.

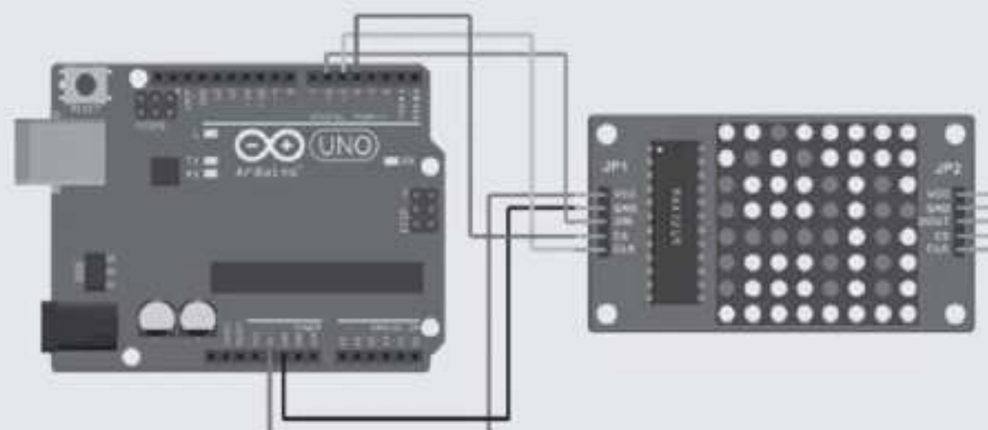


Figura 7.11 – Utilização da matriz de LEDs.

8 Sensores

Uma infinidade de tipos de sensores pode ser ligada a projetos com o Arduino e fornecer as mais variadas informações, como temperatura, umidade e variação da intensidade luminosa, entre outras. Neste capítulo mostraremos como utilizar os sensores mais populares.

8.1. Light Dependent Resistor

O LDR é um componente que varia a sua resistência conforme o nível de luminosidade que incide sobre ele. A resistência do LDR varia de forma inversamente proporcional à quantidade de luz incidente sobre ele.



PROJETO Nº 18
Uso do LDR

O objetivo deste projeto é controlar o estado de um LED (aceso ou apagado) por meio da verificação de luminosidade do ambiente utilizando um sensor de luminosidade LDR. Quanto maior for a luminosidade, menor será a resistência; por outro lado, no Arduino, maior será o valor presente na entrada analógica. Quanto menor for a luminosidade, maior será a resistência, ou seja, menor será o valor presente na entrada analógica do Arduino.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10k ohms (marrom, preto, laranja);
- 1 LED;

- 1 LDR;
- 1 protoboard;
- jumper cable.



Montagem do circuito

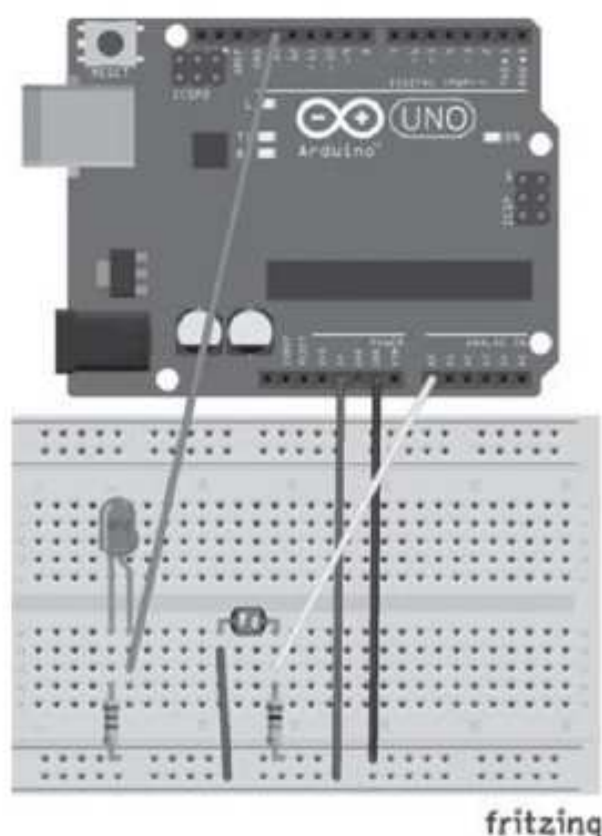


Figura 8.1 – Protótipo de iluminação de emergência.

Conforme ilustra a Figura 8.1:

a) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
c) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra linha da protoboard.
d) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor de 220 ohms;
e) Conecte o ânodo do LED ao pino 13 do Arduino.
f) Coloque o resistor de 10 kohms entre a linha de alimentação negativa e qualquer outra linha da protoboard.
g) Conecte uma das extremidades do LDR na linha do resistor de 10 k.
h) Conecte uma extremidade do jumper entre o LDR e o resistor. Conecte a outra ao pino analógico A0.
i) Conecte a outra extremidade do LDR à linha de alimentação positiva (vermelha).



Programa

No ambiente de desenvolvimento do Arduino, digite o programa a seguir:

```
int LED = 13; // Pino no qual o LED está conectado
int LDR = A0; // Pino no qual o LDR está conectado
int entrada = 0; // Variável que irá armazenar o valor do LDR

void setup() {
  pinMode(LDR, INPUT);
  pinMode(LED, OUTPUT);
}

void loop() {
  entrada = analogRead(LDR);
  delay(500);
  if (entrada < 100)
    digitalWrite(LED, HIGH); // Acende o LED
  else
    digitalWrite(LED, LOW); // Apaga o LED
  delay(100);
}
```

Para essa experiência, vamos supor o nível limite de luminosidade para que o LED se acenda como um valor menor que 100 na entrada analógica. Porém, este valor pode variar em relação ao modelo do LDR utilizado e também às condições de iluminação do ambiente. Dessa forma, se necessário, ajuste-o para obter um resultado satisfatório.



Programa

Para obter os valores gerados pela variação da resistência do LDR, seria interessante considerar exibir o valor obtido no Monitor Serial, o que facilitaria identificar o correto para a transição entre aceso e apagado. Dessa maneira, poderíamos adicionar ao nosso sketch a comunicação serial; por exemplo:

```
int LED = 13; // Pino ao qual o LED está conectado
int LDR = A0; // Pino ao qual o LDR está conectado
int entrada = 0; // Variável que irá armazenar o valor do LDR
```

```
void setup() {
Serial.begin(9600);
  pinMode(LDR, INPUT);
  pinMode(LED, OUTPUT);
}

void loop()
{ entrada = analogRead(LDR);
  Serial.println(entrada);
  delay(500);
  if (entrada < 100)
    digitalWrite(LED, HIGH); // Acende o LED
  else
    digitalWrite(LED, LOW); // Apaga o LED
  delay(100);
}
```

8.2 Sensor de temperatura



PROJETO Nº 19
Utilização do LM 35

O objetivo deste projeto é enviar os dados de um sensor de temperatura (LM 35 ou similar) para a saída serial. Na Figura 8.2 podemos ver a aparência do LM 35 que possui três pinos apresentando as respectivas funções:

Pino	Nome	Função
1	VCC	Alimentação (polo positivo)
2	Dados	Apresenta os dados de temperatura e umidade
3	GND	Alimentação (polo negativo)



Figura 8.2 – Sensor de temperatura LM 35.



Material necessário

- 1 Arduino;
- 1 LM 35 ou similar;
- 1 protoboard;
- jumper cable.



Montagem do circuito

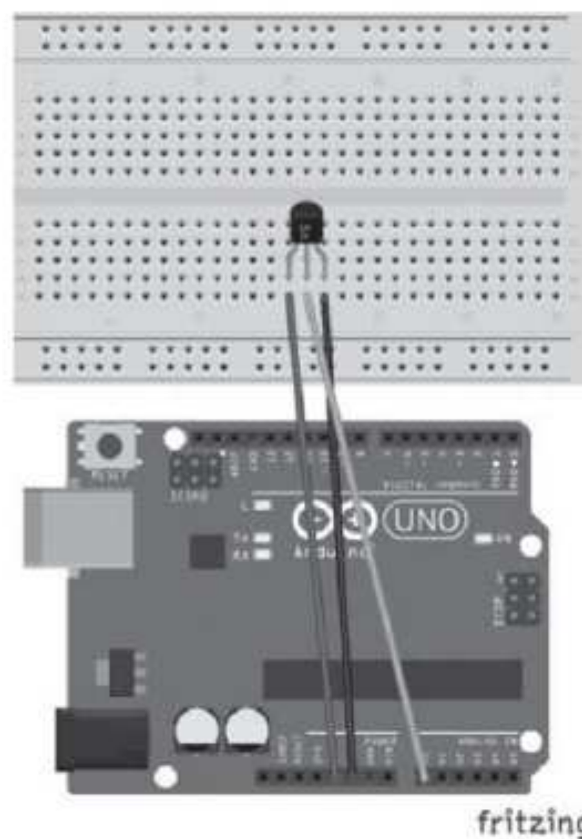


Figura 8.3 – Utilização do sensor de temperatura LM 35.

Conforme podemos observar na Figura 8.3, monte o circuito da seguinte maneira:

a) Conectar o pino 1 do LM 35 à linha de alimentação positiva (5 V) da protoboard.
b) Conectar o pino 2 do LM 35 ao pino da entrada analógica A0 do Arduino.
c) Conectar o pino 3 do LM 35 à linha de alimentação negativa (GND) da protoboard.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o seguinte sketch:

```
// Pino analógico que será ligado ao pino 2 do LM 35
const int LM 35 = A0;
```

```
//Tempo de atualização entre as leituras em ms
const int ATRASO = 5000;

//Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Temperatura: ");
  Serial.print(lerTemperatura());
  Serial.println("\260C");
  delay(ATRASO);
}

float lerTemperatura() {
  return (analogRead(LM35) * BASE_CELSIUS);
}
```

Observe no programa que realizamos a leitura da entrada analógica que é um valor entre 0 e 1.023 e depois multiplicamos pela BASE_CELSIUS, a qual consiste em uma constante para obtermos o valor em graus Celsius. Note também que, para exibirmos o símbolo de grau, utilizamos o código "\260".



Programa

Agora vamos introduzir uma pequena melhoria no programa que foi criado. Dessa maneira, utilizando as fórmulas mostradas a seguir, altere o programa para exibir a temperatura em Fahrenheit e em Kelvin.

- Conversão de Celsius para Fahrenheit: $F = (C * 9)/5 + 32$.
- Conversão de Celsius para Kelvin: $K = C + 273.15$.

No ambiente de desenvolvimento do Arduino, digite o sketch apresentado a seguir:

```
// Pino analógico que será ligado ao pino 2 do LM 35
const int LM 35 = A0;

//Tempo de atualização entre as leituras em ms
const int ATRASO = 5000;
```

```
//Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

float tempC;
float tempF;
float tempK;

void setup() {
Serial.begin(9600);
}

void loop() {
  tempC = lerTemperatura();
  tempF = (tempC * 9) / 5 + 32;
  tempK = tempC + 273.15;

  Serial.print("Temperatura em Celsius: ");
  Serial.print(tempC);
  Serial.println("\260C");

  Serial.print("Temperatura em Fahrenheit: ");
  Serial.print(tempF);
  Serial.println("\260F");

  Serial.print("Temperatura em Kelvin: ");
  Serial.print(tempK);
  Serial.println("K");

delay(ATRASO);
}

float lerTemperatura() {
  return (analogRead(LM35) * BASE_CELSIUS);
}
```

Conforme podemos observar no programa desenvolvido, a ideia básica é receber os dados do sensor, converter para graus Celsius e, em seguida, aplicar as conversões de Celsius para Fahrenheit e Celsius para Kelvin de modo a obter os demais valores. Após isso, basta enviar os respectivos valores para a saída serial.



PROJETO Nº 20
Termômetro digital simples com LM 35

Unindo os conceitos sobre displays de LCD e também sobre o uso do sensor LM 35, ambos abordados anteriormente, o objetivo deste projeto é exibir os dados obtidos por um sensor de temperatura (LM 35 ou similar) para um display de LCD.



Material necessário

- 1 Arduino;
- 1 LM 35 ou similar;
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 potenciômetro de 10 kohms;
- 1 protoboard;
- jumper cable.



Montagem do circuito

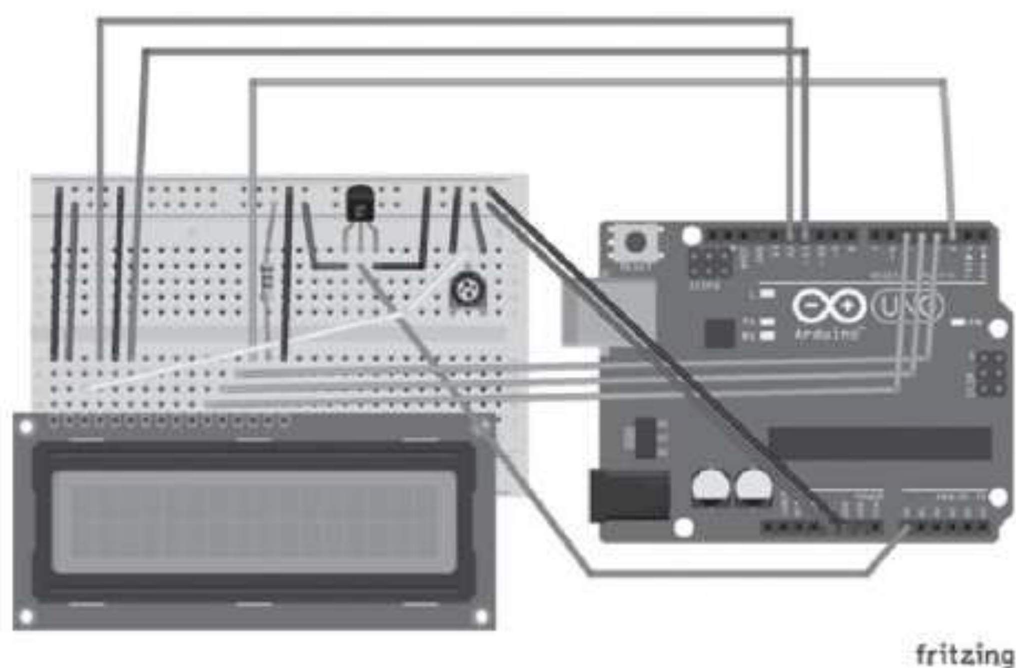


Figura 8.4 – Termômetro digital simples com LM 35.

Tomando como base a Figura 8.4, realize os seguintes passos para montar o hardware do projeto:

a) Conectar o pino 1 do LCD ligado ao GND do Arduino.

b) Pino 2 do LCD ligado ao 5 V do Arduino.

c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.
m) Conectar o pino 1 do LM 35 à linha de alimentação positiva (5 V) da protoboard.
n) Conectar o pino 2 do LM 35 ao pino da entrada analógica A0 do Arduino.
o) Conectar o pino 3 do LM 35 à linha de alimentação negativa (GND) da protoboard.



Programa

No ambiente de desenvolvimento do Arduino, digite o sketch a seguir:

```
#include <LiquidCrystal.h>

// Pino analógico do Arduino ao qual será ligado o pino 2 do LM 35
const int LM 35 = A0;

//Tempo de atualização entre as leituras em ms
const int TEMPO_ATUALIZACAO = 2000;

//Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

LiquidCrystal LCD (12, 11, 5, 4, 3, 2);
float temperatura;

void setup() {
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);
}
```

```
void loop() {
  temperatura = lerTemperatura();
  lcd.clear();
  lcd.print("Temperatura: ");
  lcd.setCursor(0, 1);
  lcd.print(temperatura);
  lcd.print("Celsius");
  delay(TEMPO_ATUALIZACAO);
}

float lerTemperatura() {
  return (analogRead(LM 35) * CELSIUS_BASE);
}
```

Conforme podemos observar neste programa, devemos utilizar a biblioteca LiquidCrystal.h para acessar as funções disponíveis para o display de LCD. Em seguida, após inicializar o display de LCD na função setup, utilizamos o método print para exibir, no display de LCD, a temperatura que foi obtida por meio do sensor de temperatura.



PROJETO Nº 21
Termômetro digital com LM 35

Vamos agora melhorar o projeto do termômetro digital construído anteriormente acrescentando um botão que permitirá selecionar as informações para serem exibidas no display de LCD. Dessa forma, o programa mostrará a temperatura não apenas em graus Celsius, mas também em Fahrenheit e Kelvin.



Material necessário

- 1 Arduino;
- 1 LM 35 ou similar;
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10 kohms (marrom, preto, laranja);
- 1 potenciômetro de 10 kohms;
- 1 push button;
- 1 protoboard;
- jumper cable.



Montagem do circuito

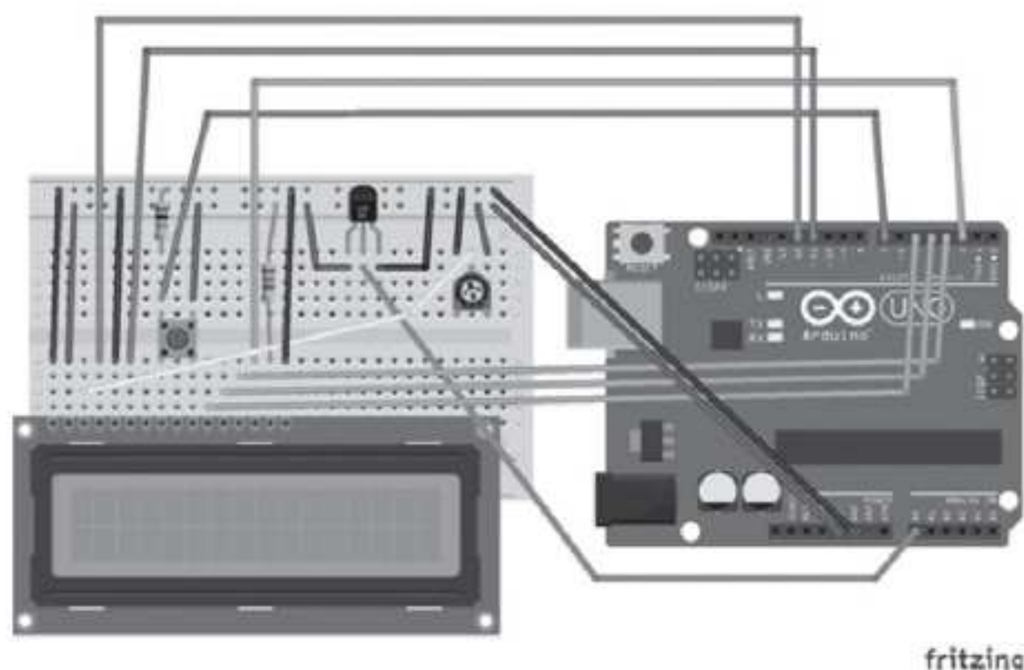


Figura 8.5 – Termômetro digital utilizando o LM 35.

Tomando como base a Figura 8.5, realize os passos a seguir para montar o hardware do projeto:

a) Conectar o pino 1 do LCD ligado ao GND do Arduino.
b) Pino 2 do LCD ligado ao 5 V do Arduino.
c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.
m) Conectar o pino 1 do LM 35 à linha de alimentação positiva (5 V) da protoboard.
n) Conectar o pino 2 do LM 35 ao pino da entrada analógica A0 do Arduino.
o) Conectar o pino 3 do LM 35 à linha de alimentação negativa (GND) da protoboard.
p) Conectar um dos pinos do botão à linha de alimentação positiva (5 V) da protoboard.
q) Conectar um dos pinos do botão ao resistor de 10 kohms.

r) Conectar o outro terminal do resistor de 10 kohms à linha de alimentação negativa (GND) da protoboard.

s) Conectar o pino do botão que está ligado ao resistor de 10 kohms à porta 7 do Arduino.



Programa

Após montar o hardware do projeto, vá até o ambiente de desenvolvimento do Arduino e digite o programa a seguir:

```
#include <LiquidCrystal.h>

// Pino analógico do Arduino ao qual será ligado o pino 2 do LM 35
const int LM 35 = A0;

// Pino de entrada do Arduino no qual o botão será conectado
const int BOTAO = 7;

//Tempo de atualização entre as leituras em ms
const int TEMPO_ATUALIZACAO = 2000;

//Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

float temperatura;
int valor;

// Opções que serão selecionadas pelo botão: 0-Celsius,
// 1-Fahrenheit, 2-Kelvin, 3-Sobre
int opcao = 0;

LiquidCrystal LCD (12, 11, 5, 4, 3, 2);

void setup() {
  opcao = 0;
  pinMode(BOTAO, INPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);
}
```

```
void loop() {
  temperatura = LerTemperatura();
  valor = digitalRead(BOTAO);
  if (valor == HIGH) {
    opcao++;
    if (opcao == 4)
      opcao = 0;
  }
  lcd.clear();
  if (opcao == 0) {
    lcd.print("Temperatura: ");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.write(B11011111); // Símbolo de grau
    lcd.print("Celsius");
  }
  else if (opcao == 1) {
    temperatura = (temperatura * 9)/ 5 + 32;
    lcd.print("Temperatura: ");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.write(B11011111); // Símbolo de grau
    lcd.print("Fahrenheit");
  }
  else if (opcao == 2) {
    temperatura = temperatura + 273.15;
    lcd.print("Temperatura: ");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.print("Kelvin");
  }
  else {
 lcd.print("TERMOMETRO, por:");
    lcd.setCursor(0, 1);
    lcd.print("Claudio e Humberto");
  }
  delay(TEMPO_ATUALIZACAO);
}

float LerTemperatura() {
  float media = 0;
  for (int i = 0; i < 10; i++)
    media += (analogRead(LM35) * BASE_CELSIUS);
  media /= 10;
  return (media);
}
```

Conforme podemos notar no trecho de código-fonte a seguir, o botão será utilizado para mudar a mensagem que será exibida no display de LCD. Cada vez que ele for pressionado, incrementamos a variável opção, até que ela tenha o valor quatro. Nesse caso, atribuímos novamente o valor inicial, que é zero.

```
valor = digitalRead(BOTAO);
if (valor == HIGH) {
  opcao++;
  if (opcao == 4)
    opcao = 0;
}
```

Em seguida, utilizamos a estrutura de if e else if para selecionar qual a informação que deverá ser mostrada no display de LCD. Ou seja, se o valor de opção for 0, mostramos a temperatura em graus Celsius; se o valor de opção for 1, exibimos a temperatura em graus Fahrenheit; se for 2, mostramos a temperatura em Kelvin e, se for igual a 3, mostramos uma informação sobre os autores do sketch.

Observe também o uso da instrução lcd.write(B11011111) para exibirmos o caractere especial que indica graus.

8.3 Sensor de temperatura e umidade



PROJETO Nº 22
Utilização do DHT-11

O objetivo deste projeto é enviar os dados de um sensor de temperatura e umidade (DHT-11 ou similar) para a saída serial.

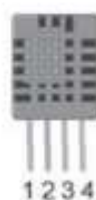


Figura 8.6 – Pinagem do sensor de temperatura DHT-11.

Na Figura 8.6 podemos ver a aparência do DHT-11, o qual possui quatro pinos que desempenham as seguintes funções:

Pino	Nome	Função
1	VCC	Alimentação (polo positivo)
2	Dados	Apresenta os dados de temperatura e umidade
3	N/C	Não conectado
4	GND	Alimentação (polo negativo)



Material necessário

- 1 Arduino;
- 1 DHT-11 ou similar;
- 1 protoboard;
- jumper cable.



Montagem do circuito

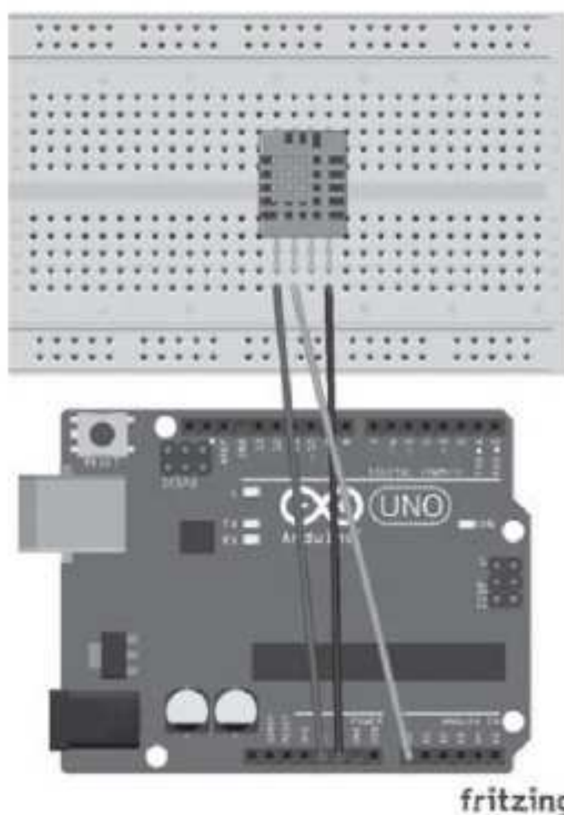


Figura 8.7 – Utilização do sensor de temperatura DHT-11.

Conforme podemos observar na Figura 8.7:

a) Conectar o pino 1 do DHT-11 à linha de alimentação positiva (5 V) da protoboard.
b) Conectar o pino 2 do DHT-11 ao pino da entrada analógica A0 do Arduino.
c) Conectar o pino 4 do DHT-11 à linha de alimentação negativa (GND) da protoboard.



Programa

Depois de montar o projeto, inicie o ambiente de desenvolvimento do Arduino e digite o seguinte sketch:

```
#include <dht.h>

// Pino analógico ao qual vai ser ligado ao pino 2 do DHT11
const int DHT11 = A0;

//Tempo de atualização entre as leituras em ms
const int ATRASO = 5000;

dht sensor;

void setup() {
  Serial.begin(9600);
}

void loop() {
  sensor.read11(DHT11); // Obtém os dados do sensor
  temperatura = sensor.temperature; // Obtém a temperatura
  umidade = sensor.humidity; // Obtém a umidade

  Serial.print("Temperatura: ");
  Serial.print(temperatura);
  Serial.print("\260C, Umidade: ");
  Serial.println(umidade);

  delay(ATRASO);
}
```

Analisando este programa, podemos observar em primeiro lugar o include para a biblioteca dht. Em seguida, criamos um objeto sensor, o qual posteriormente receberá os dados relativos à temperatura e umidade, conforme podemos ver no trecho de programa a seguir:

```
sensor.read11(DHT11); // Obtém os dados do sensor
temperatura = sensor.temperature; // Obtém a temperatura
umidade = sensor.humidity; // Obtém a umidade
```

Após a obtenção dos dados do sensor, realizaremos a exibição dos mesmos na saída serial.



PROJETO Nº 23
Termômetro digital com DHT-11

O objetivo deste projeto é enviar os dados de um sensor de temperatura e umidade (DHT-11 ou similar) para um display de LCD, no qual serão exibidas as informações.



Material necessário

- 1 Arduino;
- 1 DHT-11;
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10 kohms (marrom, preto, laranja);
- 1 potenciômetro de 10 kohms;
- 1 push button;
- 1 protoboard;
- jumper cable.



Montagem do circuito

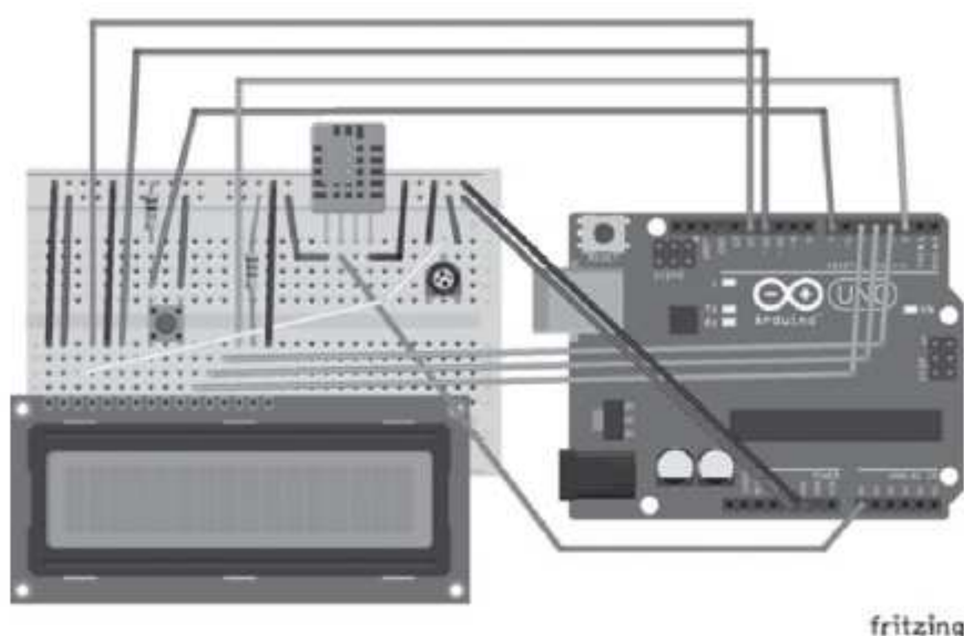


Figura 8.8 – Termômetro digital com DHT-11.

Tomando como base a Figura 8.8, realize a sequência de montagem mostrada a seguir:

a) Conectar o pino 1 do LCD ligado ao GND do Arduino.
b) Pino 2 do LCD ligado ao 5 V do Arduino.

c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.
m) Conectar o pino 1 do DHT-11 à linha de alimentação positiva (5 V) da protoboard.
n) Conectar o pino 2 do DHT-11 ao pino da entrada analógica A0 do Arduino.
o) Conectar o pino 4 do DHT-11 à linha de alimentação negativa (GND) da protoboard.
p) Conectar um dos pinos do botão à linha de alimentação positiva (5 V) da protoboard.
q) Conectar um dos pinos do botão ao resistor de 10 kohms.
r) Conectar o outro terminal do resistor de 10 kohms à linha de alimentação negativa (GND) da protoboard.
s) Conectar o pino do botão que está ligado ao resistor de 10 kohms à porta 7 do Arduino.



Programa

Digite o próximo sketch no ambiente de desenvolvimento do Arduino:

```
#include <LiquidCrystal.h>
#include <dht.h>

// Pino analógico ao qual vai ser ligado ao pino 2 do DHT11
const int DHT11 = A0;

// Pino de entrada no qual o botão está conectado
const int BOTAO = 7;

//Tempo de atualização entre as leituras em ms
const int TEMPO_ATUALIZACAO = 2000;
```

```
float temperatura;
float umidade;
int valor;

// Opções: 0-Celsius, 1-Fahrenheit, 2-Kelvin, 3-Umidade e 4-Sobre
int opcao = 0;

LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
dht sensor;

void setup() {
  opcao = 0;
  pinMode(BOTAO, INPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);
}

void loop() {
  sensor.read11(DHT11);
  temperatura = sensor.temperature;
  umidade = sensor.humidity;
  valor = digitalRead(BOTAO);
  if (valor == HIGH) {
    opcao++;
    if (opcao == 5)
      opcao = 0;
  }
  lcd.clear();
  if (opcao == 0) {
    lcd.print("Temperatura:");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.write(B11011111); // Símbolo de grau
    lcd.print("Celsius");
  }
  else if (opcao == 1) {
    temperatura = (temperatura * 9)/ 5 + 32;
    lcd.print("Temperatura:");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.write(B11011111); // Símbolo de grau
    lcd.print("Fahrenheit");
  }
```

```
  else if (opcao == 2) {
    temperatura = temperatura + 273.15;
    lcd.print("Temperatura:");
    lcd.setCursor(0, 1);
    lcd.print(temperatura);
    lcd.print("Kelvin");
  }
  else if (opcao == 3) {
    lcd.print("Umidade relativa");
    lcd.setCursor(0, 1);
    lcd.print("do ar: ");
    lcd.print(umidade);
    lcd.print(" %");
  }
  else
  { lcd.print("TERMOMETRO, por:");
    lcd.setCursor(0, 1);
    lcd.print("Claudio e Humberto");
  }
  delay(TEMPO_ATUALIZACAO);
}
```

Ao executarmos o programa criado, podemos observar que o pressionamento do botão faz com que a mensagem mostrada no display de LCD seja alterada para uma das seguintes opções: temperatura em Celsius, temperatura em Fahrenheit, temperatura em Kelvin, umidade relativa do ar e exibição de informações sobre os autores do programa.

8.4 Sensor ultrassônico



PROJETO Nº 24
Utilização do HC-SR04

O objetivo deste projeto é utilizar o sensor ultrassônico HC-SR04 para medir as distâncias entre o sensor e um objeto. O sensor HC-SR04 permite detectar objetos que lhe estão distantes entre 1 e 200 cm.



Figura 8.9 – Sensor ultrassônico HC-SR04.

Conforme podemos observar na Figura 8.9, o sensor possui quatro pinos que realizam as seguintes funções:

- VCC – Alimentação de 5 V (polo positivo).
- TRIG – Pino de gatilho (Trigger).
- ECHO – Pino de eco (Echo).
- GND – Terra (polo negativo).

Esse sensor emite um sinal ultrassônico que reflete em um objeto e retorna ao sensor, permitindo calcular a distância do objeto em relação ao sensor, adotando como base o tempo da trajetória do sinal. A velocidade do sinal no ar é de aproximadamente 340 m/s (velocidade do som). O pino ligado ao trigger (TRIG) normalmente deve estar em nível baixo. Para iniciar uma leitura de distância, o pino deve ser colocado em nível alto por 10 microssegundos e retornar para nível baixo em seguida. Nesse momento, 8 pulsos de 40 kHz são emitidos e no pino de eco (ECHO) será gerado um sinal em nível alto proporcional à distância do sensor ao objeto. Em seguida, basta verificar o tempo em que o pino ECHO permaneceu em nível alto e utilizar a fórmula do cálculo de distância (em centímetros): distância = duração/58.



Material necessário

- 1 Arduino;
- 1 sensor de ultrassom HC-SR04;
- 1 protoboard;
- jumper cable.



Montagem do circuito

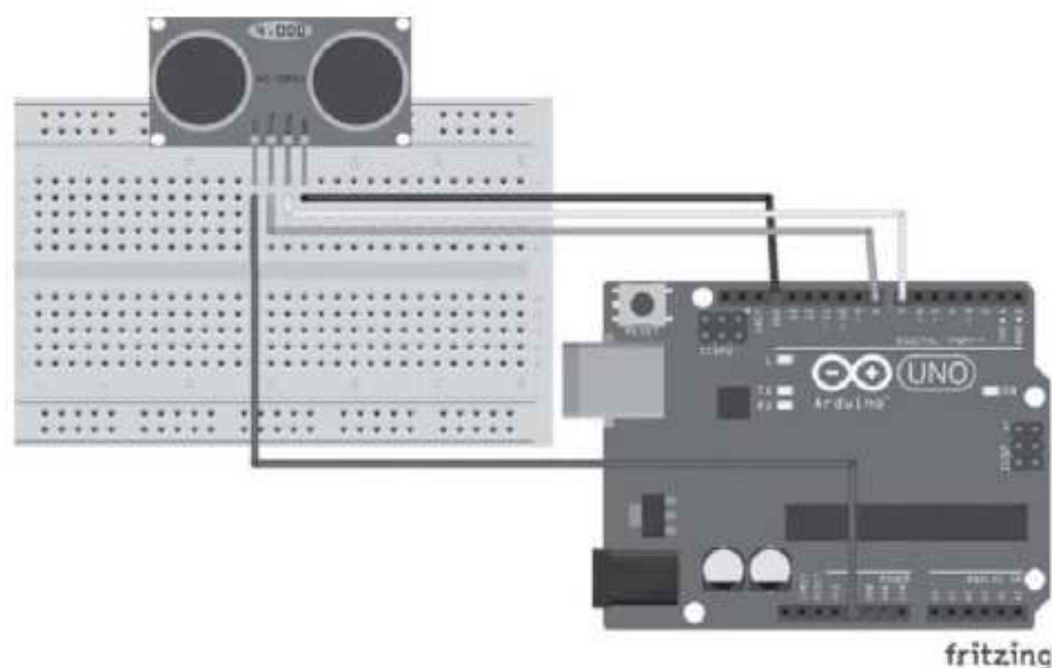


Figura 8.10 – Diagrama do circuito com o sensor de ultrassom.

Adotando como referência a Figura 8.10, realize a sequência de montagem:

a) Conectar o pino GND do HC-SR04 ligado ao GND do Arduino.
b) Conectar o pino VCC do HC-SR04 ligado ao 5 V do Arduino.
c) Conectar o pino TRIG do HC-SR04 ligado ao pino 8 do Arduino.
d) Conectar o pino ECHO do HC-SR04 ligado ao pino 7 do Arduino.



Programa

Implemente o sketch a seguir:

```
int ECHO = 7; //Pino 7 recebe o pulso
int TRIG = 8;//Pino 8 envia o pulso

long duracao = 0;
long distancia = 0;

void setup() {
Serial.begin(9600); // Inicia a porta serial
  pinMode(ECHO, INPUT); // Pino 12 é entrada (receber)
  pinMode(TRIG, OUTPUT); // Pino 13 é saída (enviar)
}

void loop() {
  // Pino trigger com um pulso baixo LOW (desligado)
  digitalWrite(TRIG, LOW);

  // Delay de 10 microssegundos
  delayMicroseconds(10);

  // Pino trigger com pulso HIGH (ligado)
  digitalWrite(TRIG, HIGH);

  // Delay de 10 microssegundos
  delayMicroseconds(10);

  // Pino trigger com um pulso baixo LOW (desligado)novamente
  digitalWrite(TRIG, LOW);
```

```
  // A função pulseInt verifica o tempo que o pino ECHO ficou HIGH
  //calculando assim a duração do tráfego do sinal
  duracao = pulseIn(ECHO,HIGH);

  // Cálculo da distância (centímetros): distância = duração/58.
  distancia = duração /58;

  Serial.print("Distancia em cm: ");
  Serial.println(distancia);
  delay(100);
}
```

Para usar o sensor devemos inicialmente mandar um sinal de gatilho (TRIGGER). Esse sinal, conforme podemos observar no trecho de programa a seguir, deverá enviar um pulso em nível 0 (LOW) por 10 microssegundos, em seguida outro em nível 1 (HIGH) durante 10 microssegundos, retornando, posteriormente, ao 0 (LOW).

```
// Pino trigger com um pulso baixo LOW (desligado)
digitalWrite(TRIG, LOW);

// Delay de 10 microssegundos
delayMicroseconds(10);

// Pino trigger com pulso HIGH (ligado)
digitalWrite(TRIG, HIGH);

// Delay de 10 microssegundos
delayMicroseconds(10);

// Pino trigger com um pulso baixo LOW (desligado) novamente
digitalWrite(TRIG, LOW);
```

Em seguida, conforme podemos visualizar no trecho de código-fonte a seguir, devemos utilizar a função pulseIn, obtendo do pino de ECHO o tempo que o sinal enviado pelo sensor, por meio do TRIGGER, levou para retornar. Dividindo esse tempo por 58, obtemos a distância em centímetros do objeto em relação ao sensor.

```
duracao = pulseIn(ECHO,HIGH);

// Cálculo da distância (centímetros): distância = duração/58.
distancia = duração /58;
```



PROJETO Nº 25
Trena digital

Utilizando os conceitos já abordados sobre o uso do display de LCD, podemos criar um sketch que permitirá exibir os dados da distância no display de LCD, tanto em centímetros quanto em polegadas. Para obter o cálculo da distância em polegadas, realize a divisão da duração do pulso por 37.



Material necessário

- 1 Arduino;
- 1 sensor de ultrassom HC-SR04;
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 potenciômetro de 10 kohms;
- 1 push button;
- 1 protoboard;
- jumper cable.



Montagem do circuito

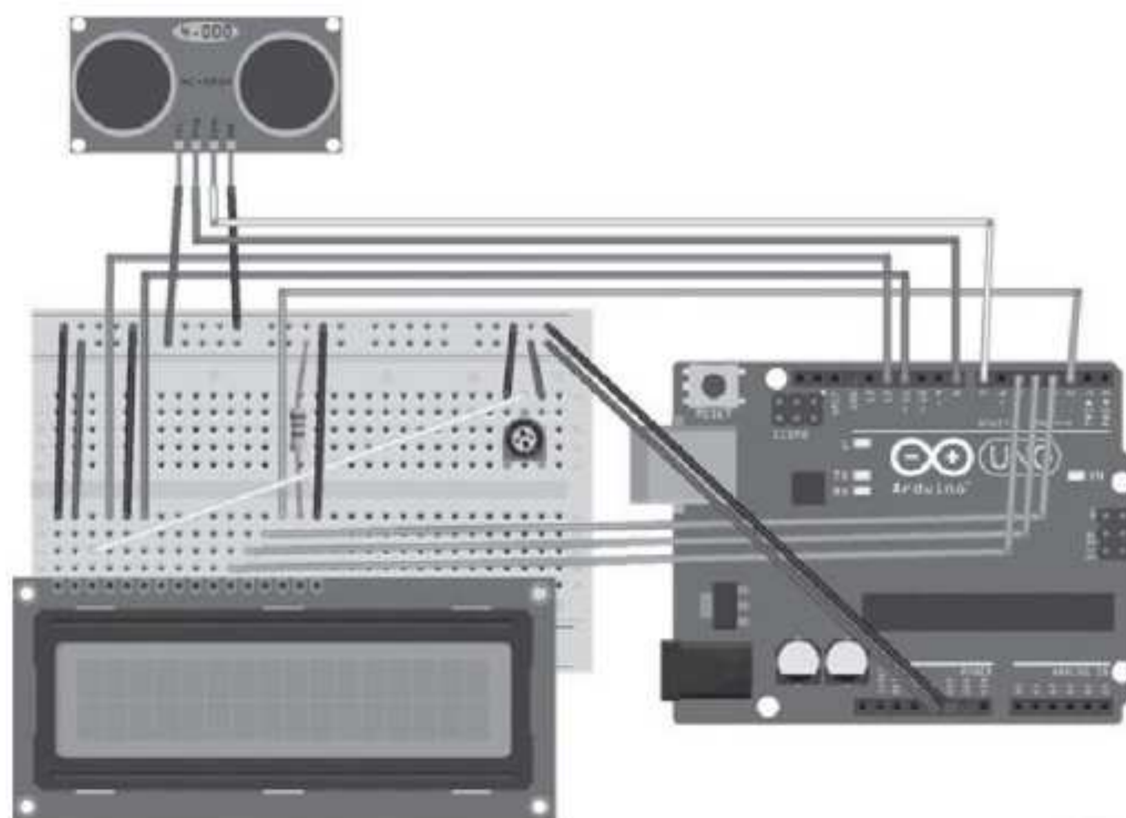


Figura 8.11 – Diagrama da trena digital.

Adotando como referência a Figura 8.11, realize os seguintes passos para montar o hardware que será usado neste projeto:

a) Conectar o pino 1 do LCD ligado à linha de alimentação negativa da protoboard.
b) Pino 2 do LCD ligado à linha de alimentação positiva (5 V) da protoboard.
c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado à linha de alimentação negativa da protoboard.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado à linha de alimentação positiva da protoboard por meio de um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD à linha de alimentação negativa da protoboard.
m) Conectar o pino GND do HC-SR04 ligado à linha de alimentação negativa da protoboard.
n) Conectar o pino VCC do HC-SR04 ligado à linha de alimentação positiva da protoboard.
o) Conectar o pino TRIG do HC-SR04 ligado ao pino 8 do Arduino.
p) Conectar o pino ECHO do HC-SR04 ligado ao pino 7 do Arduino.
q) Conectar o pino de alimentação positiva (5 V) do Arduino à linha de alimentação positiva da protoboard.
r) Conectar o pino de alimentação negativa (GND) do Arduino à linha de alimentação negativa da protoboard.



Programa

Após realizar a montagem do projeto, implemente o sketch a seguir:

```
#include <LiquidCrystal.h>

#define TEMPO_ATUALIZACAO 1000
#define ECHO 7
#define TRIG 8
```

```
int maximo = 200; // Distância máxima: 200 cm
int minimo = 0; // Distância mínima: 0 cm
long duracao, distCm, distPol;

// LCD
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);

void setup () {
pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);

  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
}

void loop () {
  // Enviar um pulso
  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  // Obter o tempo que o pulso levou para retornar
  duracao = pulseIn(ECHO, HIGH);

  // Calcular a distância em centímetros
  distCm = duracao / 58;

  // Calcular a distância em polegadas
  distPol = duracao / 37;

  lcd.clear();
  if (distCm >= maximo || distCm <= minimo) {
    lcd.print("Fora de faixa!");
  }
  else {
    lcd.print("Distancia: ");
    lcd.setCursor(0, 1); // Coluna, Linha
    lcd.print(distCm);
    lcd.print(" cm / ");
```

```
        lcd.print(distPol);
        lcd.print(" pol");
    }
    delay(TEMPO_ATUALIZACAO);
}
```

8.5 Sensor óptico reflexivo

Um sensor óptico reflexivo consiste em um diodo emissor (ou LED) de infravermelho, igual ao utilizado em controles remotos, e em um fototransistor que irá receber o sinal quando houver uma reflexão, ou seja, quando um obstáculo estiver à frente do sensor. No exemplo que iremos desenvolver, vamos utilizar o modelo TCRT 5000 (Figura 8.12), porém o projeto pode ser facilmente alterado para utilizar outro modelo similar.

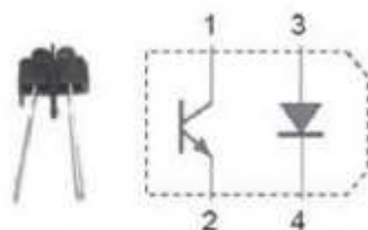


Figura 8.12 – Vistas lateral e superior do sensor óptico reflexivo TCRT 5000.

No quadro a seguir estão relacionados os pinos do sensor com as respectivas funções.

Pino	Nome	Função
1	Coletor (T+)	Coletor do fototransistor
2	Emissor (T-)	Emissor do fototransistor
3	Ânodo (D+)	Ânodo do LED infravermelho
4	Cátodo (D-)	Cátodo do LED intravermelho



PROJETO Nº 26
Interruptor de proximidade

Neste projeto vamos utilizar um sensor óptico reflexivo TCRT 5000 para implementar um interruptor de proximidade. Dessa forma, não será necessário que a pessoa toque o sensor para acender ou apagar um LED. Esse tipo de circuito é muito útil quando desejamos manter a pessoa "isolada" do circuito elétrico, evitando choques indesejáveis.



Material necessário

- 1 Arduino;
- 1 sensor óptico reflexivo TCRT 5000;
- 2 resistores de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 resistor de 10 kohms (marrom, preto, laranja);
- 1 protoboard;
- jumper cable.



Montagem do circuito

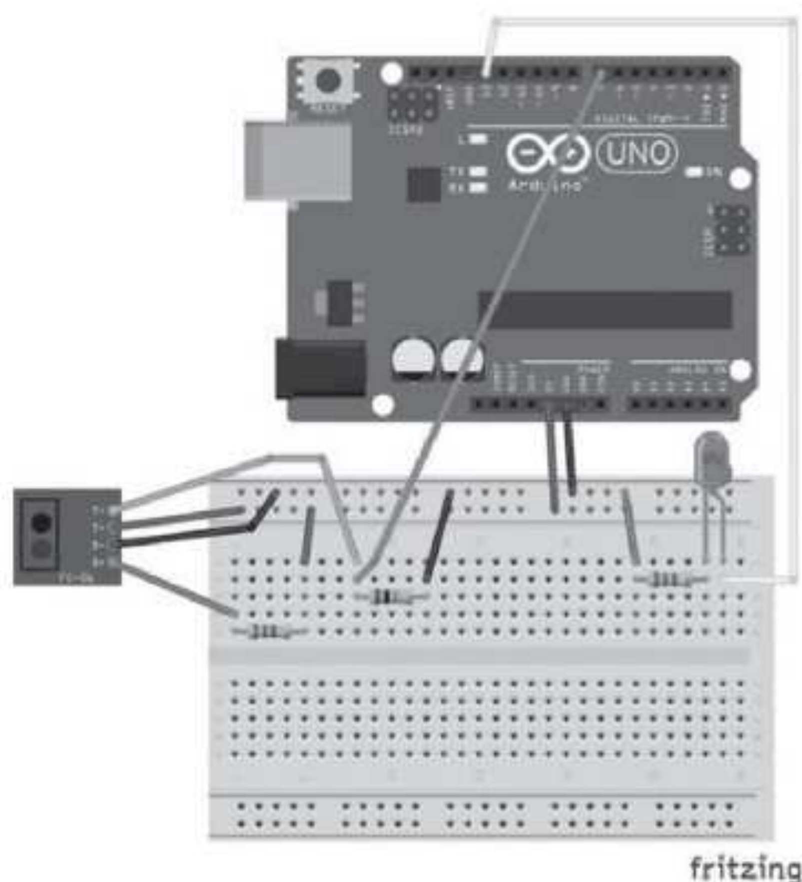


Figura 8.13 – Interruptor de proximidade.

Adotando como referência a Figura 8.13, realize os seguintes passos para montar o hardware que será usado neste projeto:

a) Insira o sensor óptico reflexivo na protoboard.
b) Conecte o pino 4 (D-) do sensor à linha de alimentação negativa (preta ou azul) da protoboard.
c) Insira um resistor de 220 ohms na protoboard e conecte um dos seus terminais ao pino 3 (D+) do sensor.

d) Conecte o outro terminal do resistor de 220 ohms à linha de alimentação positiva (vermelha) da protoboard.

e) Insira o resistor de 10 kohms na protoboard e conecte um dos seus terminais ao pino 2 (T-) do sensor.

f) Conecte o mesmo terminal do resistor de 10 kohms ao pino digital 7 do Arduino.

g) Conecte o outro terminal do resistor de 10 kohms à linha de alimentação negativa (preta ou azul) da protoboard.

h) Conecte o pino 1 (T+) do sensor à linha de alimentação positiva (vermelha) da protoboard.

i) Insira o outro resistor de 220 ohms na protoboard e conecte um dos seus terminais à linha de alimentação negativa (preta ou azul) da protoboard.

j) Insira na protoboard o LED com o cátodo (lado chanfrado e que possui o terminal mais curto) conectado ao outro terminal do resistor de 220 ohms.

k) Conecte o ânodo do LED ao pino digital 13 do Arduino.

l) Conecte o pino de 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.

m) Conecte o pino GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.



Programa

Após montar o circuito, entre no ambiente de desenvolvimento do Arduino e digite o programa a seguir.

```
int LED = 13;
int SENSOR = 7; // Pino que irá receber o sinal do fototransistor
int valor;

void setup(){
  pinMode(LED, OUTPUT);
  pinMode(SENSOR,INPUT);
}

void loop() {
  // Obter o valor do sensor
  valor = digitalRead(SENSOR);
```

```
  // Se nível HIGH não há detecção
  if (valor == HIGH) {
    // O LED será desligado
    digitalWrite(LED, LOW);
  }
  else {
    // O LED será aceso
    digitalWrite(LED,HIGH);
  }
  delay (100);
}
```

Observe neste sketch que utilizamos a função digitalRead para obter o valor do sensor. Esse valor será HIGH enquanto não houver detecção de aproximação. Quando algo se aproximar do sensor, o nível passa a ser LOW. Dessa maneira, enquanto o sensor apresentar nível alto (HIGH), mantemos o LED apagado; quando o nível presente no sensor passa a ser baixo (LOW), acendemos o LED.

Exercícios

1. Monte um circuito utilizando um sensor de temperatura (LM 35 ou DHT 11), conectado a uma entrada analógica do Arduino, e três LEDs conectados às saídas digitais do Arduino, por meio de um resistor limitador de 220 ohms (ou 330 ohms). Em seguida, crie um sketch que, quando a temperatura estiver entre 0 e 20 °C, acenda um LED, quando a temperatura estiver entre 20 e 30 °C acenda dois LEDs e quando for maior que 30 °C acenda os três.

2. Crie um sistema de estacionamento para carros (conhecidos também como parking sensor system). Esse sistema auxilia o motorista no momento de estacionar seu carro em vagas que exijam manobras em marcha a ré. Na traseira do carro existem sensores (em nosso projeto será APENAS UM SENSOR) que medem a distância de objetos e/ou carros estacionados, mostrando ao motorista a distância em centímetros, além de sinais luminosos e sonoros. O projeto deverá ter as seguintes especificações:

- Um sensor ultrassônico HC-SR04 será usado para medir a distância entre o carro e o obstáculo. Para uma visualização dessa distância, terá que ser feita uma medição em centímetros, pois terá o valor sendo exibido em um display e utilizado para conferência de vários status do sistema.
- Três LEDs representarão o status de proximidade com obstáculos: o vermelho será aceso quando o objeto estiver a uma distância menor do que 5 cm do sensor; o amarelo, quando o obstáculo estiver em uma distância maior ou igual a 5 cm até igual a 10 cm de distância do sensor; e o verde aceso indicará uma distância ainda segura, acima de 10 cm de distância do sensor.

- Um buzzer dará os sinais sonoros de proximidade. Quanto mais perto do obstáculo, mais curto será o intervalo dos bipes emitidos pelo buzzer. Por exemplo, se o obstáculo estiver a 10 cm de distância, serão emitidos bipes a cada 1.000 ms (emite um bipe – aguarda 1.000 ms – emite um bipe – aguarda 1.000 ms –...). O intervalo dos bipes irá diminuir quanto mais próximo o obstáculo estiver do sensor. Se a distância for de 9 cm, o intervalo será de 900 ms; se for 8 cm será de 800 ms, e assim por diante. Quanto menor o intervalo da emissão de som, mais alerta o motorista terá que ficar, pois indicará uma possível colisão. Esse sinal sonoro só se iniciará quando a distância for menor ou igual a 10 cm.
- Será utilizado um display LCD 16x2 para mostrar todos os estados do sistema. Quando o LED verde estiver aceso, o display mostrará "SEGURO: XX cm". Caso o amarelo esteja aceso, o display terá que mostrar "ATENÇÃO: XX cm". E quando o vermelho estiver aceso e a distância for maior que 2 cm, mostrará "CUIDADO: XX cm". Por fim, se o LED vermelho estiver aceso e a distância for menor ou igual a 2 cm, o display mostrará "PARE!!!".

3. Desenvolva um projeto que controle a luminosidade de uma estufa por sensores e controles de abertura e fechamento de telas que controlam a entrada de luz solar em um ambiente simulado. Essa estufa foi projetada para plantas ornamentais, que precisam de baixa luminosidade para seu crescimento ideal. Antes da construção real da estufa, é necessário criar uma base de teste. Para essa base de teste, são definidos os seguintes parâmetros:

- Um LDR será utilizado para verificar a luminosidade do local. A partir dele deverá ser avaliada a luminosidade mais alta possível do local (o valor mais alto alcançado). Esse valor tem que ser verificado por meio do Serial Monitor durante algum tempo e será usado como referência (valor máximo de luminosidade do local). Por exemplo, se o maior valor captado pelo LDR for 835, esse será o máximo (referente a 100% de luminosidade);
- Três LEDs representarão o status de luminosidade do local: o LED vermelho será aceso quando a luminosidade do local estiver entre 80% a 100% da luminosidade máxima do local; o amarelo será aceso quando a luminosidade estiver entre 50 a 79% da luminosidade máxima do local; e o verde indica uma luminosidade ideal, abaixo dos 50%;
- Um buzzer irá representar o controle das telas. Quando o LED estiver vermelho, as 3 telas de proteção deverão ser fechadas, para impedir a entrada de luz solar e assim reduzir a luminosidade. Com isso, o buzzer deve apitar três sinais breves, sendo cada um deles indicativo de que uma tela está fechada. Se o LED estiver amarelo, serão emitidos 2 apitos breves (2 telas fechadas). Quando o LED estiver verde, será ouvido 1 apito (1 tela fechada);
- O sistema também terá um botão de liga/desliga. Durante o dia (período que o sistema deverá estar ligado), é necessário clicar uma vez no botão (push button). Pressionar novamente o botão indica o desligamento do sistema, ou seja, o sistema não estará mais ativo.

9 Módulos

O conceito de módulos facilita a montagem de projetos que, antes do Arduino, exigiam conhecimentos avançados em eletrônica. Dessa forma, podemos entender que os módulos consistem em circuitos eletrônicos, na maioria das vezes bastante complexos, que realizam uma função específica e podem ser facilmente conectados ao Arduino e, posteriormente, programados por meio de bibliotecas específicas.

9.1 Relógio em tempo real



PROJETO Nº 27
Relógio com LCD

O objetivo deste projeto é criar um relógio digital a partir de um módulo Real Time Clock (RTC) e um display LCD 16x2. Neste projeto usaremos as bibliotecas RTClib.h e LiquidCrystal.h. A RTClib.h, disponível para download em https://github.com/adafruit/RTClib, fornecerá as funções necessárias para a utilização do módulo RTC DS-1307. Por outro lado, a LiquidCrystal.h possui funções que auxiliam nas configurações e no tratamento dos dados a serem enviados ao LCD. A montagem do display deve ser de acordo com a sua especificação, com cada um dos pinos apresentando uma função específica.



Material necessário

- 1 Arduino;
- 1 RTC (DS-1307);
- 1 LCD 16x2;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);

- 1 potenciômetro de 10 kohm;
- 1 protoboard;
- jumper cable.



Montagem do circuito

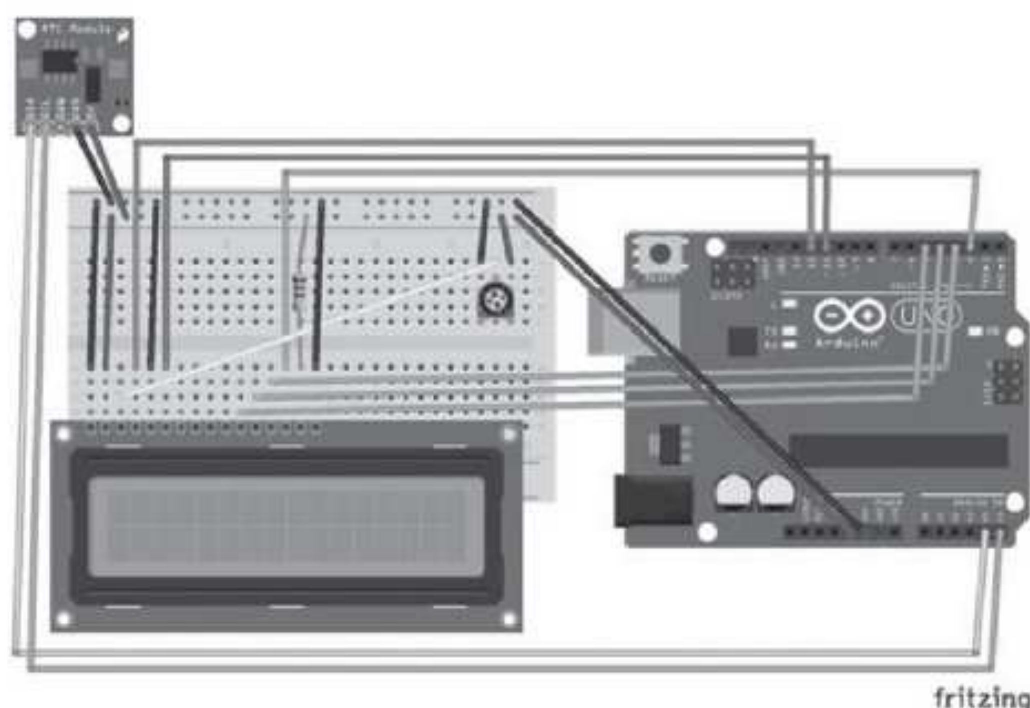


Figura 9.1 – Relógio digital com LCD.

Utilizando a Figura 9.1 como referência, realize os passos a seguir para a montagem:

a) Pino 1 do LCD ligado ao GND do Arduino.
b) Pino 2 do LCD ligado ao 5 V do Arduino.
c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.
m) Pino SDA do RTC ligado ao pino analógico A4 do Arduino.
n) Pino SCL do RTC ligado ao pino analógico A5 do Arduino.

o) Pino GND do RTC ligado ao pino GND do Arduino.

p) Pino VCC do RTC ligado ao 5 V do Arduino.



Programa

Antes de montar o sketch deste projeto, devemos notar que a biblioteca RTCLib não faz parte da distribuição padrão da IDE do Arduino. Dessa forma, torna-se necessário importá-la antes de utilizar o RTC pela primeira vez. Para isso, no menu “Sketch”, escolha a opção “Importar Biblioteca” e “Add Library”, conforme indicado pela Figura 9.2.



Figura 9.2 – Adicionando uma biblioteca ao Arduino IDE.

Em seguida, conforme ilustra a Figura 9.3, selecione o arquivo compactado (ZIP) que contém a biblioteca a ser importada, ou seja:

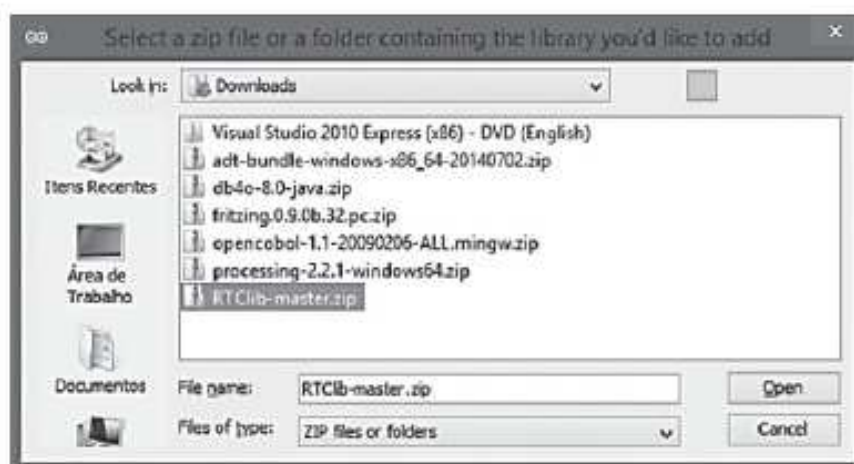


Figura 9.3 – Selecionando o arquivo.

Após importar a biblioteca, ela estará disponível para uso dentro da opção do menu “Importar Biblioteca”, conforme podemos observar na Figura 9.4.

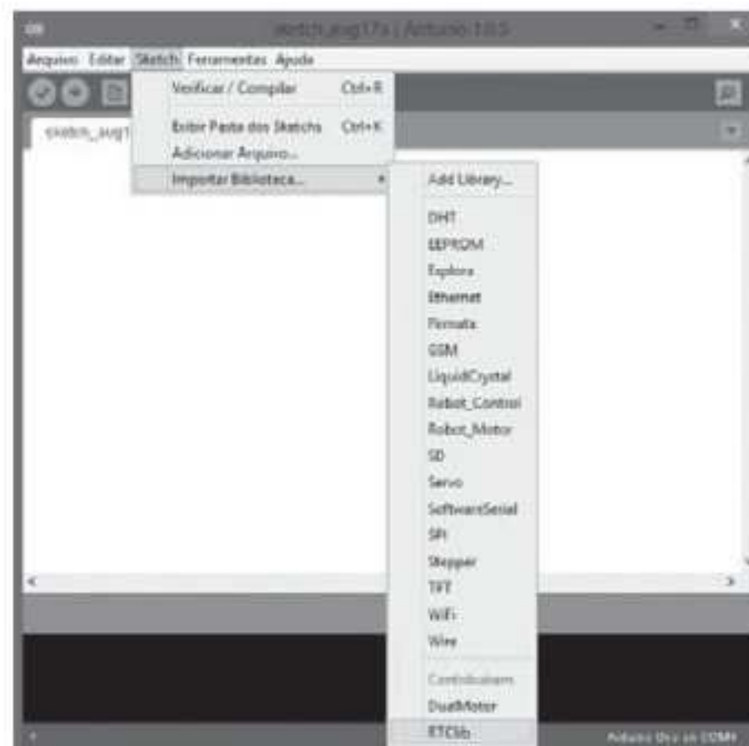


Figura 9.4 – Bibliotecas instaladas.

Após a instalação da biblioteca, digite o programa a seguir:

```
#include <Wire.h>
#include <RTClib.h>
#include <LiquidCrystal.h>

#define TEMPO_ATUALIZACAO 1000

// RTC - Real Time Clock
// Conectar SCL (RTC) em A5 (Arduino) e SDA (RTC) em A4 (Arduino).

RTC_DS1307 RTC;

// Pinagem do ACM 1602K
// Pin Símbolo Função                   Conectar
//  1  Vss     Alimentação (-)          GND
//  2  Vdd     Alimentação (+5V)        VCC
//  3  Vo      Ajuste de contraste      Potenciômetro
//  4  RS      Seleção     Arduino 12
//  5  R/W     Leitura/Escrita          GND
//  6  E       Habilitar a escrita      Arduino 11
//  7  DB0     Bit de dados 0           NC
//  8  DB1     Bit de dados 1           NC
//  9  DB2     Bit de dados 2           NC
// 10  DB3     Bit de dados 3           NC
// 11  DB4     Bit de dados 4           Arduino 5
// 12  DB5     Bit de dados 5           Arduino 4
// 13  DB6     Bit de dados 6           Arduino 3
```

```
// 14  DB7     Bit de dados 7       Arduino 2
// +   BL+     Alimentação BL+      Resistor de 1 k para VCC
// -   BL-     Alimentação BL-      GND

LiquidCrystal LCD (12, 11, 5, 4, 3, 2);

int dia, mês, ano, hora, minuto, segundo, dia_semana;
char semana[][4] = {"DOM", "SEG", "TER", "QUA", "QUI",
  "SEX", "SAB"};

void setup () {
  Serial.begin(9600);
  Wire.begin();  // Inicialização do protocolo Wire
                 // (necessário para que o módulo RTC funcione)

  RTC.begin(); // Inicialização do módulo RTC
  // Verifica se o módulo está funcionando
  if (! RTC.isrunning()) {
    Serial.println("O RTC não está executando!");
  }

  // Ajusta o relógio com a data e hora na qual o programa,
  // foi compilado, necessário apenas quando é preciso
  // ajustar a data e a hora do RTC
  // RTC.adjust(DateTime(__DATE__, __TIME__));

  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);
}

void loop () {
  DateTime now = RTC.now(); //Recuperando a data e hora atual
  dia = now.day();
  mes = now.month();
  ano = now.year();
  hora = now.hour();
  minuto = now.minute();
  segundo = now.second();
  dia_semana = now.dayOfWeek();

  lcd.clear();
  if (dia < 10)
    lcd.print("0");
```

```
  lcd.print(dia, DEC);
  lcd.print("/");
  if (mes < 10)
    lcd.print("0");
  lcd.print(mes, DEC);
  lcd.print("/");
  lcd.print(now.year(), DEC);
  lcd.setCursor(13, 0); // Coluna, Linha
  lcd.print(semana[dia_semana]);
  lcd.setCursor(0, 1); // Coluna, Linha
  if (hora < 10)
    lcd.print("0");
  lcd.print(hora, DEC);
  lcd.print(":");
  if (minuto < 10)
    lcd.print("0");
  lcd.print(minuto, DEC);
  lcd.print(":");
  if (segundo < 10)
    lcd.print("0");
  lcd.print(segundo, DEC);
  delay(TEMPO_ATUALIZACAO);
}
```

Analisando este programa, podemos observar que, inicialmente, declaramos um objeto para o relógio:

```
RTC_DS1307 RTC;
```

Conforme podemos observar no trecho do sketch a seguir, na função setup devemos inicializar primeiro o protocolo Wire que é requerido pela biblioteca do RTC. Em seguida, o módulo do RTC deve ser inicializado pelo método begin. Note também que quando for necessário ajustar data e hora armazenadas de RTC, usaremos o método adjust.

```
void setup () {
  Wire.begin(); // Inicialização do protocolo Wire
                // (necessário para que o módulo RTC funcione)

  RTC.begin();  // Inicialização do módulo RTC
  // Verifica se o módulo está funcionando
  if (! RTC.isrunning()) {
    Serial.println("O RTC não está executando!");
  }
```

```
  // Ajusta o relógio com a data e hora na qual o programa,
  // foi compilado, necessário apenas quando é preciso
  // ajustar a data e a hora do RTC
  // RTC.adjust(DateTime(__DATE__, __TIME__));

// Continuação do programa...
}
```

Olhando o trecho de programa a seguir, podemos observar que, na função loop, usamos o método now para obter a data e a hora que estão armazenadas no módulo RTC e as armazenamos em um objeto da classe DateTime. Em seguida, são usados os métodos day, month, year, hour, minute, second e dayOfWeek para obtermos cada um dos componentes de data e hora em variáveis inteiras.

```
DateTime now = RTC.now(); //Recuperando a data e hora atual
dia = now.day();
mes = now.month();
ano = now.year();
hora = now.hour();
minuto = now.minute();
segundo = now.second();
dia_semana = now.dayOfWeek();
```

Após obtermos data e hora, devemos realizar a sua exibição no display de LCD. Por exemplo, a seguir, podemos observar como o valor da variável 'dia' será mostrado. Observe que para mantermos a formatação de dois dígitos, quando o valor da variável for menor que dez, devemos exibir um zero antes.

```
lcd.clear();
if (dia < 10)
  lcd.print("0");
lcd.print(dia, DEC);
```

O vetor semana será aplicado para mostrar o nome abreviado do dia da semana que foi obtido por meio do método dayOfWeek, ou seja:

```
lcd.print(semana[dia_semana]);
```



PROJETO Nº 28
Relógio com termômetro

Neste projeto vamos utilizar os conceitos abordados anteriormente sobre sensores de temperatura e adicioná-los ao projeto do relógio digital.



Material necessário

- 1 Arduino;
- 1 RTC (DS-1307);
- 1 LCD 16x2;
- 1 sensor de temperatura LM 35 ou similar;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 potenciômetro de 10 kohm;
- 1 protoboard;
- jumper cable.



Montagem do circuito

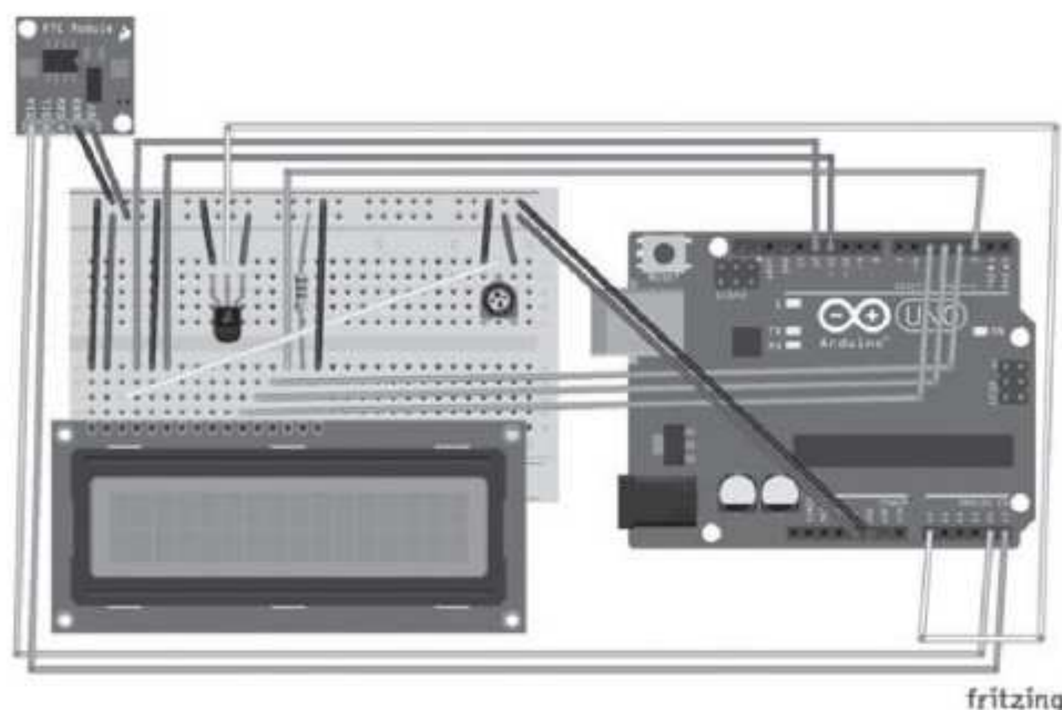


Figura 9.5 – Relógio com termômetro.

Tomando como referência a Figura 9.5, realize a sequência de montagem:

a) Pino 1 do LCD ligado ao GND do Arduino.
b) Pino 2 do LCD ligado ao 5 V do Arduino.

c) Pino 3 do LCD ligado ao pino central do potenciômetro (controle de contraste).
d) Pino 4 do LCD ligado ao pino digital 12 do Arduino.
e) Pino 5 do LCD ligado ao GND do Arduino.
f) Pino 6 do LCD ligado ao pino digital 11 do Arduino.
g) Pino 11 do LCD ligado ao pino digital 5 do Arduino.
h) Pino 12 do LCD ligado ao pino digital 4 do Arduino.
i) Pino 13 do LCD ligado ao pino digital 3 do Arduino.
j) Pino 14 do LCD ligado ao pino digital 2 do Arduino.
k) Pino 15 do LCD ligado ao 5 V do Arduino com um resistor de 220 ohms (ou 330 ohms).
l) Pino 16 do LCD ligado ao GND do Arduino.
m) Pino SDA do RTC ligado ao pino analógico A4 do Arduino.
n) Pino SCL do RTC ligado ao pino analógico A5 do Arduino.
o) Pino GND do RTC ligado ao pino GND do Arduino.
p) Pino VCC do RTC ligado ao 5 V do Arduino.
q) Conectar o pino 1 do LM 35 à linha de alimentação positiva (5 V) da protoboard.
r) Conectar o pino 2 do LM 35 ao pino da entrada analógica A0 do Arduino.
s) Conectar o pino 3 do LM 35 à linha de alimentação negativa (GND) da protoboard.



Programa

Como podemos observar no código-fonte a seguir, o programa é bem parecido com aquele que foi desenvolvido no projeto anterior, apenas acrescentamos o sensor de temperatura e depois exibimos o valor da temperatura, obtido pelo sensor, no display de LCD, juntamente com os dados de data e hora. No ambiente de desenvolvimento do Arduino, digite o seguinte sketch:

```
#include <Wire.h>
#include <RTClib.h>
#include <LiquidCrystal.h>

#define TEMPO_ATUALIZACAO 1000

// RTC - Real Time Clock
// Conectar SCL (RTC) em A5 (Arduino) e SDA (RTC) em A4 (Arduino)

RTC_DS1307 RTC;
```

```
// Pinagem do ACM 1602K
// Pin Símbolo Função              Conectar
//  1  Vss     Alimentação (-)     GND
//  2  Vdd     Alimentação (+5V)   VCC
//  3  Vo      Ajuste de contraste Potenciômetro
//  4  RS      Seleção    Arduino  12
//  5  R/W     Leitura/Escrita     GND
//  6  E       Habilitar a escrita Arduino 11
//  7  DB0     Bit de dados 0      NC
//  8  DB1     Bit de dados 1      NC
//  9  DB2     Bit de dados 2      NC
// 10  DB3     Bit de dados 3      NC
// 11  DB4     Bit de dados 4      Arduino 5
// 12  DB5     Bit de dados 5      Arduino 4
// 13  DB6     Bit de dados 6      Arduino 3
// 14  DB7     Bit de dados 7      Arduino 2
// +   BL+     Alimentação BL+     Resistor de 1 k para VCC
// -   BL-     Alimentação BL-     GND

LiquidCrystal LCD (12, 11, 5, 4, 3, 2);

// Pino analógico do Arduino ao qual será ligado ao pino 2 do LM 35
const int LM 35 = A0;

//Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

int dia, mês, ano, hora, minuto, segundo, dia_semana;
char semana[][4] = {"DOM", "SEG", "TER", "QUA", "QUI", "SEX", "SAB"};

void setup () {
  Serial.begin(9600); // Inicialização da comunicação serial
  Wire.begin(); // Inicialização do protocolo Wire
  RTC.begin(); // Inicialização do módulo RTC
  // Verifica se o módulo está funcionando ou não
  if (! RTC.isrunning()) {
    Serial.println("O RTC não está executando!");
  }

  // Ajusta o relógio com a data e hora na qual
  // o programa foi compilado, retirar o comentário da linha
  // de baixo apenas quando for necessário ajustar a data e a
// hora do RTC
// RTC.adjust(DateTime(__DATE__, __TIME__));
```

```
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  lcd.begin (16, 2);
}

void loop () {
  DateTime now = RTC.now(); // Recuperando a data e hora atual
  dia = now.day();
  mes = now.month();
  ano = now.year();
  hora = now.hour();
  minuto = now.minute();
  segundo = now.second();
  dia_semana = now.dayOfWeek();

  lcd.clear();
  if (dia < 10)
    lcd.print("0");
  lcd.print(dia, DEC);
  lcd.print("/");
  if (mes < 10)
    lcd.print("0");
  lcd.print(mes, DEC);
  lcd.print("/");
  lcd.print(now.year(), DEC);
  lcd.setCursor(13, 0); // Coluna, Linha
  lcd.print(semana[dia_semana]);
  lcd.setCursor(0, 1); // Coluna, Linha
  if (hora < 10)
  lcd.print("0");
  lcd.print(hora, DEC);
  lcd.print(":");
  if (minuto < 10)
    lcd.print("0");
  lcd.print(minuto, DEC);
  lcd.print(":");
  if (segundo < 10)
    lcd.print("0");
  lcd.print(segundo, DEC);
```

```
  // Exibir a temperatura
  lcd.setCursor(10, 1); // Coluna, Linha
  lcd.print (lerTemperatura(), DEC);
  lcd.setCursor(14, 1); // Coluna, Linha
  lcd.write(B11011111); // Símbolo de grau
  lcd.print ("C");

  delay(TEMPO_ATUALIZACAO);
}

float lerTemperatura() {
  return (analogRead(LM35) * BASE_CELSIUS);
}
```

No trecho de programa a seguir, podemos observar a exibição da temperatura no display de LCD:

```
// Exibir a temperatura
lcd.setCursor(10, 1); // Coluna, Linha
lcd.print (lerTemperatura(), DEC);
lcd.setCursor(14, 1); // Coluna, Linha
lcd.write(B11011111); // Símbolo de grau
lcd.print ("C");
```



PROJETO Nº 29
Relógio com displays de LEDs

Neste projeto vamos exibir a hora obtida por meio do módulo RTC em um display de LEDs de sete segmentos e quatro dígitos, também abordado anteriormente.



Material necessário

- 1 Arduino;
- 1 RTC (DS-1307);
- 1 display de LED de sete segmentos (quatro dígitos);
- 1 CI MAX 7219 ou 7221;
- 1 resistor de 100 kohms (marrom, preto, amarelo);
- 1 protoboard;
- jumper cable.



Montagem do circuito

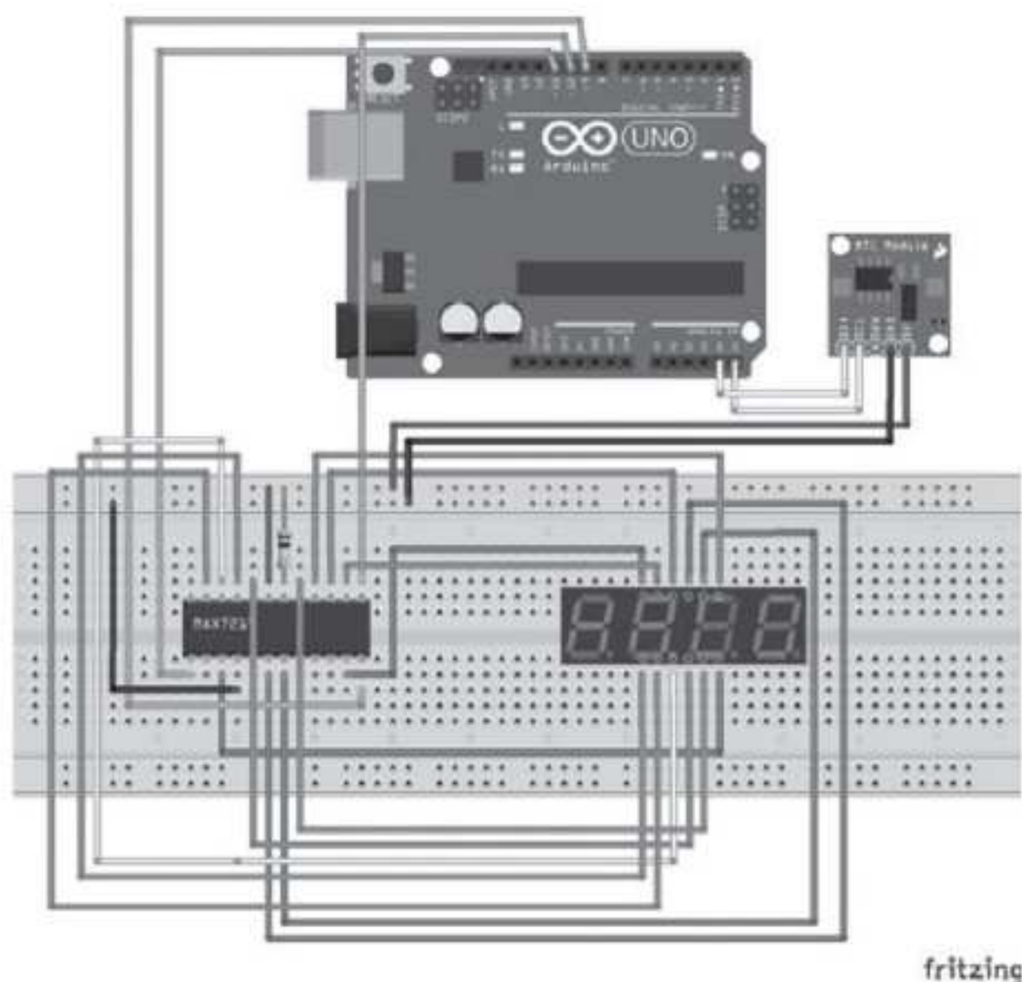


Figura 9.6 – Relógio com displays de LEDs de sete segmentos.

Conforme ilustrado no diagrama mostrado na Figura 9.6, realize os passos necessários para a montagem do projeto:

a) Conecte o pino 1 (DIN) do MAX 7219/21 ao pino 11 do Arduino.
b) Conecte o pino 3 (DIG 4) do MAX 7219/21 ao pino 6 do display.
c) Conecte o pino 4 (GND) do MAX 7219/21 à linha de alimentação negativa da protoboard.
d) Conecte o pino 6 (DIG 2) do MAX 7219/21 ao pino 9 do display.
e) Conecte o pino 7 (DIG 3) do MAX 7219/21 ao pino 8 do display.
f) Conecte o pino 11 (DIG 1) do MAX 7219/21 ao pino 12 do display.
g) Conecte o pino 12 (LOAD/CS) do MAX 7219/21 ao pino 9 do Arduino.
h) Conecte o pino 13 (CLK) do MAX 7219/21 ao pino 10 do Arduino.
i) Conecte o pino 14 (SEG A) do MAX 7219/21 ao pino 11 do display.
j) Conecte o pino 15 (SEG F) do MAX 7219/21 ao pino 10 do display.
k) Conecte o pino 16 (SEG B) do MAX 7219/21 ao pino 7 do display.
l) Conecte o pino 17 (SEG G) do MAX 7219/21 ao pino 5 do display.

m) Conecte o pino 18 (SEG F) do MAX 7219/21 ao resistor de 100 kohms.

n) Conecte o outro terminal do resistor à linha de alimentação positiva (5 V) da protoboard.

o) Conecte o pino 19 (V+) do MAX 7219/21 à linha de alimentação positiva (5 V) da protoboard.

p) Conecte o pino 20 (SEG C) do MAX 7219/21 ao pino 4 do display.

q) Conecte o pino 21 (SEG E) do MAX 7219/21 ao pino 1 do display.

r) Conecte o pino 22 (SEG DP) do MAX 7219/21 ao pino 13 da matriz.

s) Conecte o pino 23 (SEG D) do MAX 7219/21 ao pino 6 da matriz.

t) Conecte o pino 5 V do RTC à linha de alimentação positiva da protoboard.

u) Conecte o pino GND do RTC à linha de alimentação negativa da protoboard.

v) Conecte o pino SDA do RTC à entrada analógica A4 do Arduino.

w) Conecte o pino SCL do RTC à entrada analógica A5 do Arduino.

x) Conecte o pino 5 V do Arduino à linha de alimentação positiva da protoboard.

y) Conecte o pino GND do Arduino à linha de alimentação negativa da protoboard.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o sketch (programa) a seguir:

```
#include <Wire.h>
#include <RTClib.h>
#include "LEDControl.h"

// RTC - Real Time Clock
// Conectar SCL (RTC) em A5 (Arduino) e SDA (RTC) em A4 (Arduino)
RTC_DS1307 RTC;

/*
 * Criar um LEDControl (lc).
 * O pino 11 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 10 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 9 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino
*/
LEDControl lc = LEDControl(11, 10, 9, 1);
```

```
int dia, mês, ano, hora, minuto, segundo;
char sHora[3], sMinuto[3];
boolean ponto = false;

void setup() {
  Serial.begin(9600);
  Wire.begin(); // Inicialização do protocolo Wire
  RTC.begin(); // Inicialização do módulo RTC
  // Verifica se o módulo está funcionando ou não
  if (! RTC.isrunning()) {
 Serial.println("O RTC não está executando!");
  }

  // Ajusta o relógio com a data e hora na qual o programa foi
  // compilado
  // RTC.adjust(DateTime(__DATE__, __TIME__));

  // Retira o MAX 7219/21 do modo de economia de energia
  lc.shutdown(0, false);

  // Define a intendidade luminosa do display
lc.setIntensity(0, 3);
}

void loop() {
  // Recuperando a data e hora atual
  DateTime now = RTC.now();

  dia = now.day();
  mes = now.month();
  ano = now.year();
  hora = now.hour();
  minuto = now.minute();
  segundo = now.second();

  Serial.print("Data: ");
  Serial.print(dia);
  Serial.print("/");
  Serial.print(mes);
  Serial.print("/");
  Serial.print(ano);
  Serial.print(" - Hora: ");
  Serial.print(hora);
  Serial.print(":");
  Serial.print(minuto);
```

```
  Serial.print(":");
  Serial.println(segundo);

  // Converte a variável inteira hora para String
  dtostrf(hora, 2, 0, sHora);

  // Converte a variável inteira minuto para String
  dtostrf(minuto, 2, 0, sMinuto);

  if (hora < 10)
    lc.setChar(0, 1, '0', false);
  else
    lc.setChar(0, 1, sHora[0], false);
  lc.setChar(0, 2, sHora[1], ponto);
  if (minuto < 10)
    lc.setChar(0, 3, '0', false);
  else
    lc.setChar(0, 3, sMinuto[0], false);
  lc.setChar(0, 4, sMinuto[1], false);
  ponto = !ponto;
  delay(1000);
}
```

Quando utilizamos os displays de LEDs, devemos enviar individualmente cada caractere (ou dígito) a ser exibido. Além disso, devemos manter a formatação de dois dígitos para hora e minuto. Assim, conforme podemos observar no fragmento de programa a seguir, vamos utilizar a função dtostrf para converter a hora e minuto de um valor inteiro a uma cadeia de caracteres (string). Em seguida, verificamos se a hora é menor do que dez; caso afirmativo, exibimos por meio do método setChar o caractere '0'; caso contrário, mostramos o valor que está armazenado na primeira posição da string, ou seja, sHora[0]. A seguir é exibido o valor que está na segunda posição da string, isto é, sHora[1].

```
dtostrf(hora, 2, 0, sHora);
dtostrf(minuto, 2, 0, sMinuto);
if (hora < 10)
  lc.setChar(0, 1, '0', false);
else
  lc.setChar(0, 1, sHora[0], false);
lc.setChar(0, 2, sHora[1], ponto);
if (minuto < 10)
  lc.setChar(0, 3, '0', false);
else
  lc.setChar(0, 3, sMinuto[0], false);
lc.setChar(0, 4, sMinuto[1], false);
```

Ainda nesse trecho de programa, podemos notar que o mesmo processo descrito para a exibição da hora é repetido para mostrar dois dígitos que compõem os minutos.

9.2 Bluetooth

O Bluetooth permite a comunicação sem fio de dispositivos que estejam próximos, até no máximo 10 m. É uma forma de comunicação de dados bastante eficiente, de baixo custo e que possibilita que o seu projeto possa se comunicar com dispositivos como tablets e smartphones, entre outros. No Arduino temos vários modelos de módulos ou shields que podem ser utilizados e possibilitam a comunicação por Bluetooth. Nos exemplos utilizados neste livro nos basearemos no módulo HC-05, bastante comum e de custo acessível.



Figura 9.7 – Módulo Bluetooth HC-05.

Conforme podemos observar na Figura 9.7, o HC-05 apresenta 6 pinos, sendo que os KEY e STATE são utilizados para a configuração do módulo e não serão utilizados nos projetos que iremos desenvolver, uma vez que o módulo já vem previamente configurado e não é necessário alterar os parâmetros. Dessa forma, a ligação do módulo ao Arduino é muito simples: realizamos a ligação da alimentação (5 V e GND) e depois conectamos do pino de transmissão do módulo (TX) ao de recepção (RX) do Arduino, ou seja, o pino digital 0. Em seguida, conectamos o pino de recepção do módulo (RX) ao de transmissão (TX) do Arduino, que é o pino digital 1.



PROJETO Nº 30
Bluetooth LED

Neste projeto vamos realizar a comunicação Bluetooth de modo a permitir que um LED seja aceso ou apagado remotamente por outro dispositivo.



Material necessário

- 1 Arduino;
- 1 módulo Bluetooth HC-05 ou similar;
- 1 LED;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

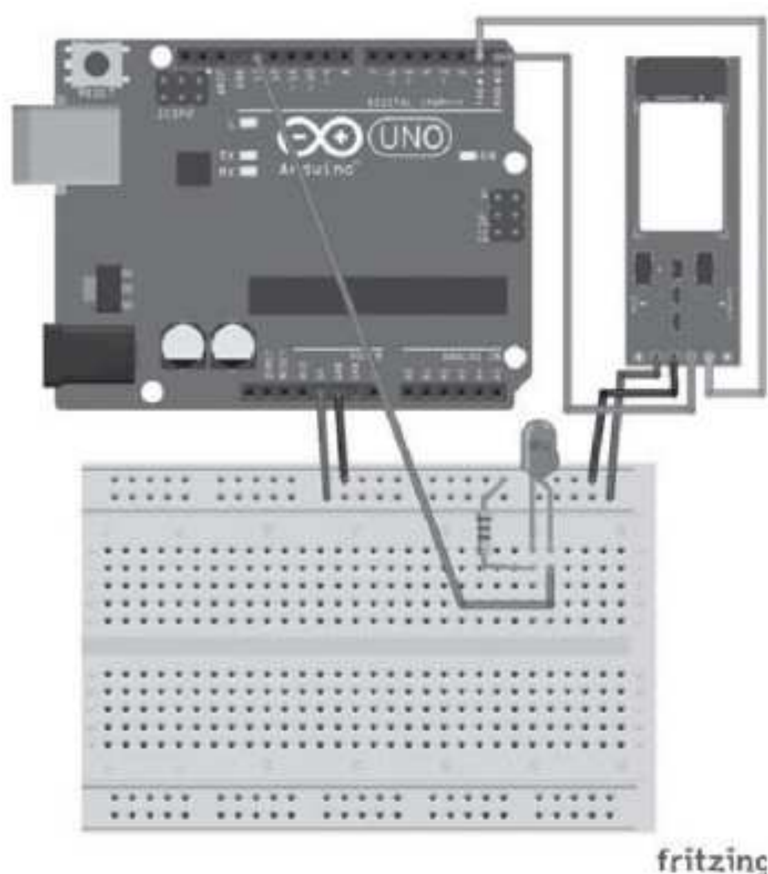


Figura 9.8 – Diagrama do projeto Bluetooth LED.

Considerando como referência a Figura 9.8, realize a sequência de montagem:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.

b) Conecte o pino de 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.

c) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.

d) Coloque o LED com o cátodo (lado chanfrado) conectado ao terminal do resistor;
e) Conecte o ânodo do LED ao pino 13 do Arduino.
f) Conecte o pino GND do módulo Bluetooth à linha de alimentação negativa da protoboard.
g) Conecte o pino VCC do módulo Bluetooth à linha de alimentação positiva da protoboard.
h) Conecte o pino de transmissão (TX) do módulo Bluetooth ao pino digital 0 (RX) do Arduino.
i) Conecte o pino de recepção (RX) do módulo Bluetooth ao pino digital 1 (TX) do Arduino.



Programa

Como podemos observar no sketch a seguir, a implementação é bastante simples, visto que o módulo Bluetooth utilizará a comunicação serial:

```
int LED = 13;

void setup() {
  Serial.begin (9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  if (Serial.available()) {
char caractere = (char) Serial.read();
if(caractere == '1') {
      digitalWrite(LED, HIGH);
      Serial.println("LED ligado.");
    }
    else if (caractere == '0') {
      digitalWrite(LED, LOW);
      Serial.println("LED desligado.");
}
  }
  delay(1000);
}
```

Analisando o código-fonte, podemos observar que, quando o caractere '1' é recebido pelo módulo, o LED será aceso; quando o '0' for recebido, o LED será apagado e qualquer outro caractere será ignorado.

Para testar o programa, devemos inicialmente "parear" o dispositivo que utilizaremos (computador, tablet ou mesmo um smartphone) com o módulo HC-05 e em seguida utilizar um terminal para envio dos comandos, que, neste exemplo, consistem apenas nos caracteres '0' e '1'. A seguir demonstraremos esses passos utilizando um computador com o sistema operacional Windows.

Ative o Bluetooth no computador e clique sobre o ícone de Dispositivos Bluetooth que será mostrado na Barra de Tarefas, conforme podemos observar na Figura 9.9.



Figura 9.9 – Dispositivos Bluetooth.

Em seguida será exibido um menu. Escolha a opção "Mostrar Dispositivos Bluetooth" e a janela mostrada na Figura 9.10 será aberta. Aguarde até que o dispositivo HC-05 (ou outro similar) seja mostrado. Quando isso ocorrer, clique no botão "Emparelhar".

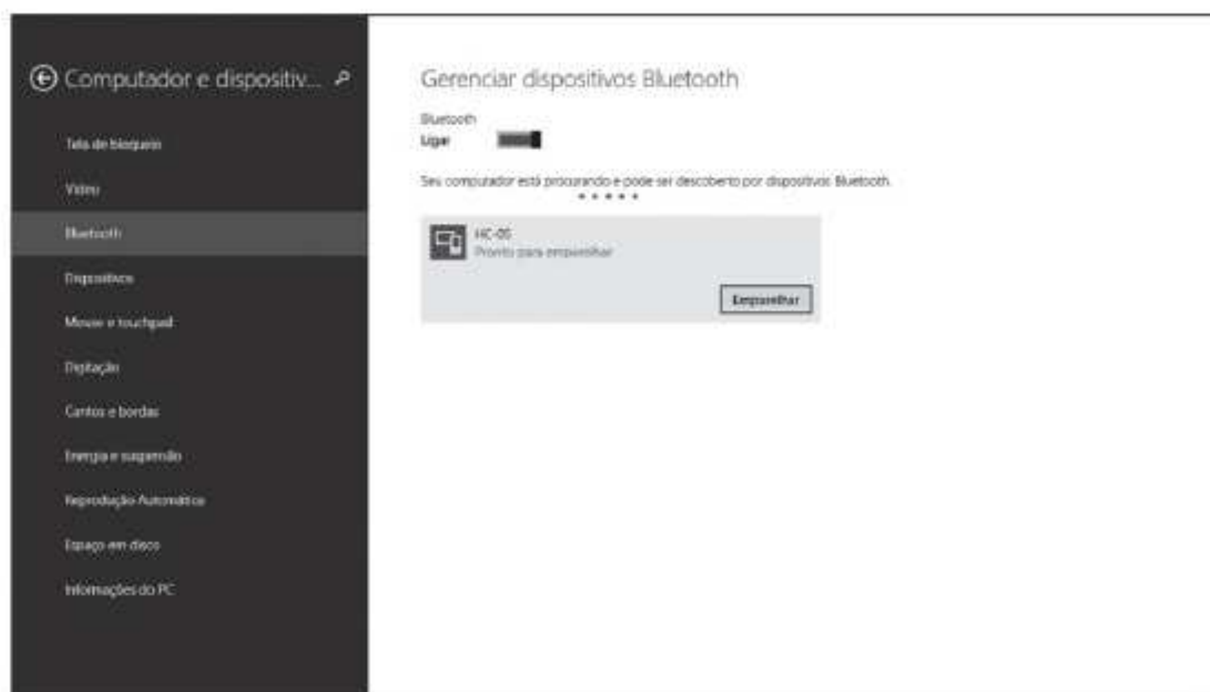


Figura 9.10 – Gerenciamento dos dispositivos Bluetooth.

Será solicitada a senha para emparelhar o HC-05 (ou similar). Digite 1234, que é a senha padrão, conforme ilustra a Figura 9.11.



Figura 9.11 – Emparelhar um dispositivo.

Após a instalação do dispositivo, devemos identificar quais portas estão em uso para que seja possível realizarmos a conexão. Para fazer isso, clique novamente no ícone Bluetooth que está na "Barra de Tarefas" e, em seguida, selecione o item do menu "Abrir Configurações". Conforme ilustrado na Figura 9.12, na janela que foi aberta, selecione a aba "Portas COM" e identifique aquelas que estão sendo utilizadas para comunicação com o módulo Bluetooth.

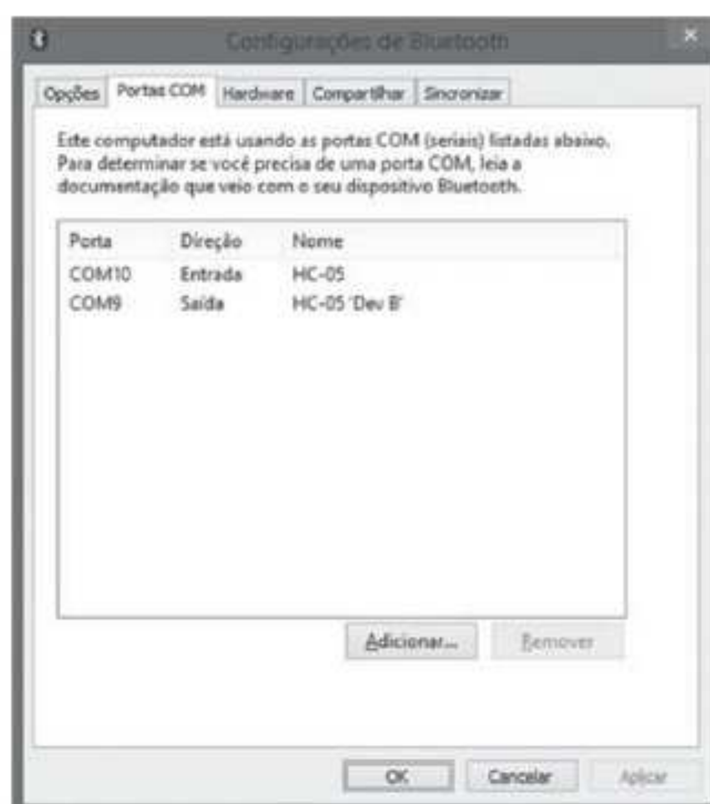


Figura 9.12 – Portas utilizadas pelo dispositivo Bluetooth.

Agora utilize um programa emulador de terminal, como o PuTTY, que é gratuito e está disponível para download em http://www.putty.org/, para enviar os comandos por meio do Bluetooth. Após abrir o programa, selecione a porta de saída e faça a conexão conforme ilustrado na Figura 9.13.

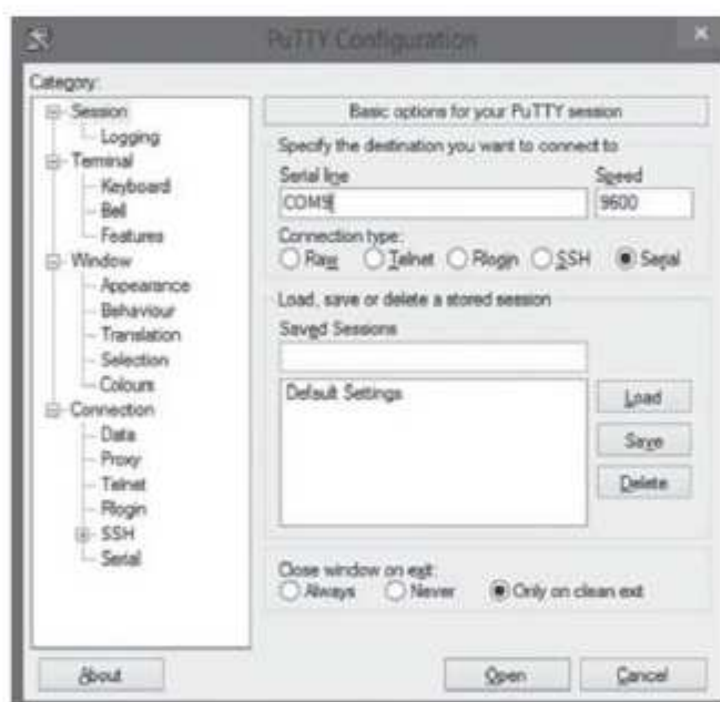


Figura 9.13 – Conexão por meio do PuTTY ao dispositivo Bluetooth.

Após o terminal ser mostrado (Figura 9.14), digite o caractere 1 seguido da tecla Enter e o LED será aceso. Quando digitar o caractere 0 seguido de um Enter, o LED deverá ser apagado.



Figura 9.14 – Envio de dados para o dispositivo Bluetooth.

Outra opção é utilizar o próprio Monitor Serial do ambiente de desenvolvimento do Arduino, apenas tomando a precaução de mudar a porta serial para aquela que está sendo utilizada pelo Bluetooth em vez da porta que é utilizada pelo Arduino.

9.3 Controle remoto com infravermelho

Existem diversos kits, similares ao mostrado na Figura 9.15, que possuem um receptor de infravermelho que pode receber sinais de um controle remoto. Para utilizar a recepção e transmissão de infravermelho, será necessária a utilização da biblioteca IRremote, que pode ser baixada gratuitamente no endereço https://github.com/shirriff/Arduino-IRremote.



Figura 9.15 – Kit com controle remoto e receptor infravermelho.



PROJETO Nº 31
Controle de um LED por controle remoto

Neste projeto vamos acender ou apagar um LED pelas teclas 1 ou 0 de um controle remoto. Utilizaremos um receptor de infravermelho similar ao mostrado na Figura 9.16.



Figura 9.16 – Receptor infravermelho.

O pino 1 do receptor é o de sinal, o 2 é a alimentação positiva (5 V), enquanto o 3 é a alimentação negativa (GND). Se você estiver utilizando um modelo diferente, identifique corretamente a função dos pinos antes de conectá-los ao restante do circuito.



Material necessário

- 1 Arduino;
- 1 receptor de infravermelho;
- 1 transmissor de infravermelho (controle remoto);
- 1 LED;

- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

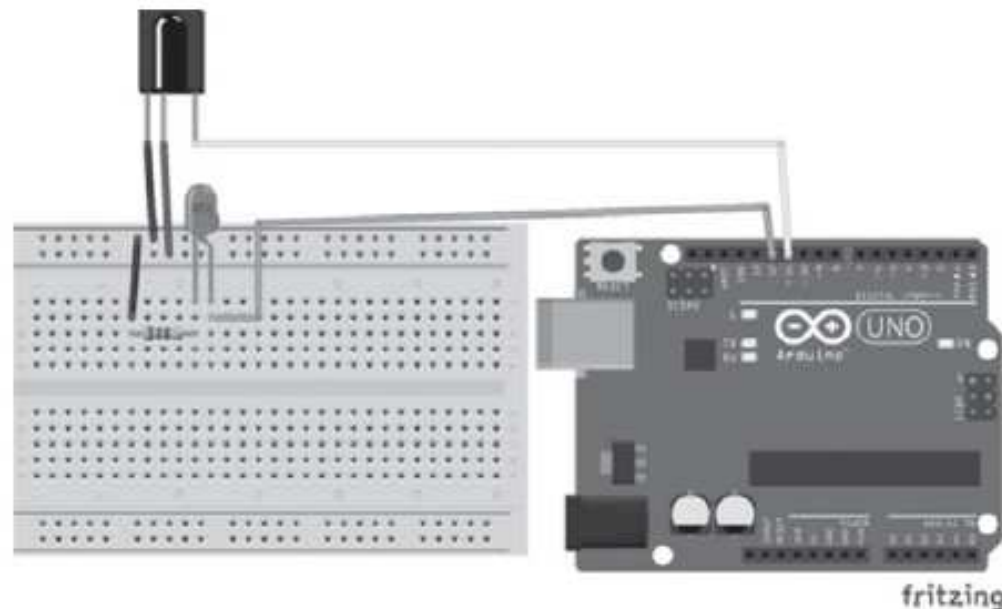


Figura 9.17 – Conexão do receptor infravermelho ao Arduino.

Considerando como referência a Figura 9.17, realize a sequência de montagem:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
c) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
d) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
e) Conecte o ânodo do LED ao pino 12 do Arduino.
f) Pino GND do receptor infravermelho na linha de alimentação GND da protoboard.
g) Pino VCC do receptor infravermelho na linha de alimentação positiva (5 V) da protoboard.
h) Pino de dados do receptor infravermelho no pino 11 do Arduino.



Programa

Após realizar o download da biblioteca IRremote, copie os arquivos IRremote.h, IRremote.cpp e IRremoteInt.h para a pasta do seu sketch; então, digite o código-fonte a seguir:

```
#include "IRremote.h"

int RECEPTOR = 11;
int LED = 12;
int estado = LOW;

IRrecv controle(RECEPTOR);
decode_results resultado;

void setup() {
  Serial.begin(9600);

  // Iniciar a recepção
  controle.enableIRIn();

  // O LED 13 piscará sempre que um sinal for recebido
  controle.blink13(true);

  pinMode(LED, OUTPUT);
}

void loop() {
  if (controle.decode(&resultado)) {
    Serial.println(resultado.value, HEX);

    if (resultado.value == 0xFF6897)      // Tecla 0
      estado = LOW;
    else if (resultado.value == 0xFF30CF) // Tecla 1
      estado = HIGH;

    digitalWrite(LED, estado);

    // Obter o próximo valor
    controle.resume();
  }
}
```

Analisando o trecho de código-fonte a seguir, podemos notar que, inicialmente, devemos declarar o pino, no qual o receptor de infravermelho está conectado ao Arduino, como um objeto da classe IRrecv que fará a recepção e também uma estrutura que receberá os dados (decode_results) enviados pelo controle remoto ao receptor.

```
int RECEPTOR = 11;

IRrecv controle(RECEPTOR);
decode_results resultado;
```

Conforme podemos visualizar no código-fonte a seguir, para verificar se uma tecla do controle remoto foi pressionada, utilizamos o método decode e na estrutura 'resultado' teremos as informações que foram recebidas.

```
if (controle.decode(&resultado)) {
  Serial.println(resultado.value, HEX);

  if (resultado.value == 0xFF6897)      // Tecla 0
    estado = LOW;
  else if (resultado.value == 0xFF30CF) // Tecla 1
    estado = HIGH;

  digitalWrite(LED, estado);

  // Obter o próximo valor
  controle.resume();
}
```

Utilize a saída serial para obter o valor em hexadecimal relativo às teclas que você deseja utilizar no projeto. Note que esses valores podem variar em relação ao modelo de controle remoto utilizado. Neste exemplo, mapeamos o valor 0xFF6897 para a tecla "0" e 0xFF30CF para a "1". Se necessário, ajuste esses valores para o controle remoto que você possui. Também observe, neste mesmo exemplo, que, para possibilitar a recepção dos dados do próximo acionamento do controle remoto, devemos utilizar o método resume.

Além do código da tecla acionada, a estrutura decode_results também traz outras informações sobre o controle remoto utilizado. Dessa forma, é possível empregarmos o atributo decode_type para identificar o modelo do controle remoto, conforme visto no trecho de programa a seguir.

```
if (controle.decode(&resultado)) {
  Serial.println(resultado.value, HEX);

  if (resultado.decode_type == NEC) {
    Serial.print("NEC: ");
  }
  else if (resultado.decode_type == SONY) {
    Serial.print("SONY: ");
  }
```

```
  else if (resultado.decode_type == RC5) {
    Serial.print("RC5: ");
  }
  else if (resultado.decode_type == RC6) {
    Serial.print("RC6: ");
  }
  else if (resultado.decode_type == UNKNOWN) {
    Serial.print("Desconhecido ");
  }

  if (resultado.value == 0xFF6897)       // Tecla 0
    estado = LOW;
  else if (resultado.value == 0xFF30CF) // Tecla 1
    estado = HIGH;

  digitalWrite(LED, estado);

  // Obter o próximo valor
  controle.resume();
}
```

Exercícios

1. Elabore um projeto que utilize o controle remoto infravermelho para acender ou apagar individualmente cada componente de cor de um LED RGB. A tecla 1 do controle remoto deverá acender ou apagar a cor vermelha, a 2 deverá acender ou apagar a cor verde, enquanto a 3 acenderá ou apagará o LED azul. A tecla 0 deverá apagar completamente o LED.

2. Altere o Projeto Nº 28 (relógio com termômetro) de modo a utilizar o controle remoto infravermelho para realizar a seleção da unidade de temperatura a ser mostrada (Celsius, Fahrenheit ou Kelvin). As teclas + e – do controle remoto deverão ser utilizadas para realizar a troca.

10 Motores e Servomotores

Os motores e servomotores são atuadores que permitem a construção de projetos nos quais podemos dar movimento a uma plataforma robótica ou permitir o voo de um drone, entre outras possibilidades.

10.1 Motores de corrente contínua

O princípio básico de funcionamento dos motores de corrente contínua (Figura 10.1) consiste em deixar a corrente elétrica fluir por uma bobina, criando um campo magnético. Esse campo magnético aplicado a um ímã resultará no giro de um eixo, o qual poderá estar conectado a rodas, hélices ou qualquer outro tipo de engrenagem.



Figura 10.1 – Exemplo do motor de corrente contínua.

Os motores costumam ter um consumo elevado de corrente elétrica. Assim, se ligados diretamente às portas do Arduino, podem ocasionar danos a essas portas. Dessa forma, utilizamos circuitos que permitem "isolar" um motor da porta do Arduino que irá controlar o funcionamento do motor.



PROJETO Nº 32
Controle básico de um motor

O objetivo deste projeto é demonstrar o acionamento de um motor de corrente contínua por meio de uma porta analógica do Arduino. O circuito que irá acionar o motor irá utilizar um diodo e um transistor. O diodo, como visto, possui dois terminais, o ânodo e o cátodo, sendo que o cátodo possui uma faixa em seu corpo, conforme indica a seta na Figura 10.2.



Figura 10.2 – Diodo.

O transistor utilizado serão os modelos BD 135, 137 ou 139, conforme ilustrado na Figura 10.3. Esse componente, quando olhado de frente, apresenta os seguintes terminais: 1 – Emissor (E), 2 – Coletor (C) e 3 – Base (B).



Figura 10.3 – Transistor.



Material necessário

- 1 Arduino;
- 1 transistor BD 135, 137 ou 139;
- 1 diodo 1N 4001 (ou 1N 4004);
- 1 motor de corrente contínua de 4,5 ou 6,0 V;
- 1 resistor de 1 kohm (marrom, preto, vermelho);
- 1 suporte para três (4,5 V) ou quatro pilhas (6,0 V);
- 1 protoboard;
- jumper cable.



Montagem do circuito

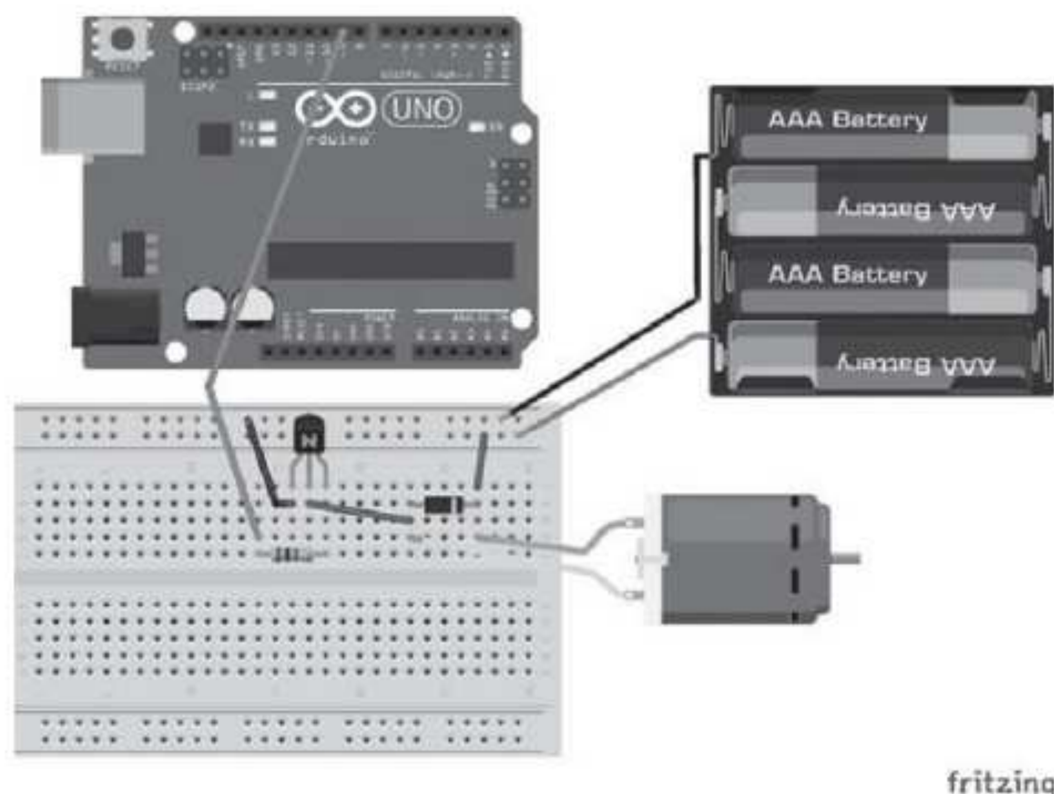


Figura 10.4 – Controle básico de um motor.

Quando montar esse circuito, tome muito cuidado com as ligações do diodo e do transistor, pois se conectados de forma errada poderão ser danificados. Assim, considerando como referência a Figura 10.4, realize a sequência de montagem:

a) Insira o transistor na protoboard.

b) Conecte o pino 1 do transistor (Emissor) na linha de alimentação negativa (preta ou azul) da protoboard.

c) Insira o resistor de 1 kohms na protoboard conectando um dos seus terminais ao pino 3 (Base) do transistor.

d) Conecte o outro terminal do resistor de 1 kohms à porta de saída 9 do Arduino.

e) Insira os conectores do motor na protoboard, conectando um dos terminais ao pino 2 (Coletor) do transistor.

f) Conecte o outro terminal do motor à linha de alimentação positiva (vermelha) da protoboard.

g) Insira o diodo na protoboard, conectando o cátodo (terminal identificado por uma faixa no corpo do componente) na linha de alimentação positiva (vermelha) da protoboard.

h) Conecte o ânodo do diodo ao outro terminal do motor, o qual já foi conectado ao pino 2 (Coletor) do transistor.

i) Conecte o polo negativo (–) do suporte para as pilhas à linha de alimentação negativa (preto ou azul) da protoboard.

j) Conecte o polo positivo (+) do suporte para as pilhas à linha de alimentação positiva (vermelha) da protoboard.

No projeto montado, observe o uso do transistor para isolar e amplificar (ampliar) o sinal da porta de saída do Arduino, evitando, dessa forma, danos a ela. Também note o uso do diodo para evitar que uma inversão de polos, gerada pela bobina do motor devido ao efeito indutivo, cause danos ao restante do circuito.



Programa

Após montar o circuito e verificar as conexões realizadas, digite o código-fonte a seguir no ambiente de desenvolvimento do Arduino.

```
// Controle básico de um motor de corrente contínua

int MOTOR = 9;
int velocidade;

void setup() {
  pinMode(MOTOR, OUTPUT);
}

void loop() {
  // Aumentar a velocidade de rotação do motor
  for (velocidade = 0 ; velocidade < 256; velocidade++) {
    analogWrite(MOTOR, velocidade);
    delay(25);
  }

  // Manter o motor desligado por um segundo
  analogWrite(MOTOR, 0);
  delay(1000);
}
```

O funcionamento do sketch é bem simples: definimos o pino onde está conectado o motor como saída, que, neste exemplo, é o 9. Em seguida utilizamos um laço de repetição for para gerar um valor entre zero e 255, que produzirá a aceleração gradativa do motor. A função analogWrite colocará esse valor no pino de saída ao qual o motor está conectado.

Dessa maneira, note que o valor zero representa o motor parado e 255 é a velocidade máxima de rotação do motor. Em seguida, o motor é desligado por um segundo, e, após esse tempo, o ciclo inicia-se novamente.



PROJETO Nº 33
Controle da velocidade do motor

Neste projeto vamos utilizar um potenciômetro, conectado a uma entrada analógica do Arduino, para ajustar a velocidade de rotação de um motor de corrente contínua.



Material necessário

- 1 Arduino;
- 1 transistor BD 135, 137 ou 139;
- 1 diodo 1N 4001 (ou 1N 4004);
- 1 motor de corrente contínua de 4,5 ou 6,0 V;
- 1 resistor de 1 kohms (marrom, preto, vermelho);
- 1 potenciômetro de 10 kohms;
- 1 suporte para três (4,5 V) ou quatro pilhas (6,0 V);
- 1 protoboard;
- jumper cable.



Montagem do circuito

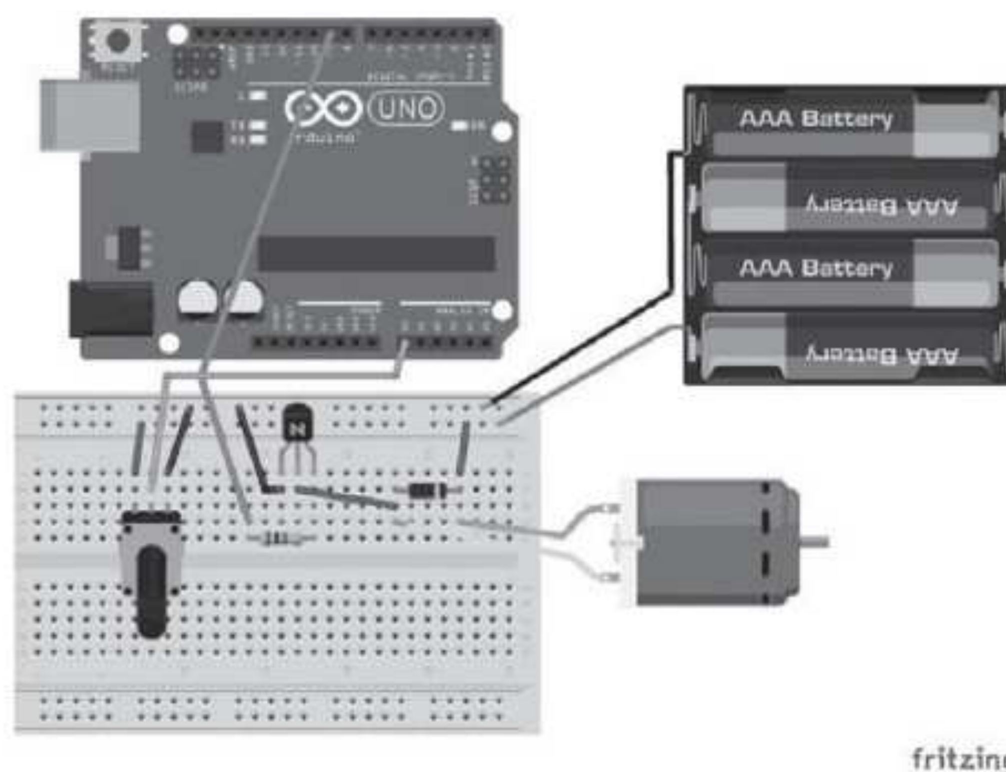


Figura 10.5 – Controle do motor pelo potenciômetro.

Quando montar este circuito, tome bastante cuidado para não inverter as ligações dos terminais do diodo e transistor, pois, se conectados de forma errada, poderão ser danificados. Assim, adotando como referência a Figura 10.5, realize a sequência de montagem:

a) Insira o transistor na protoboard.
b) Conecte o pino 1 do transistor (Emissor) à linha de alimentação negativa (preta ou azul) da protoboard.
c) Insira o resistor de 1 kohm na protoboard conectando um dos seus terminais no pino 3 (Base) do transistor.
d) Conecte o outro terminal do resistor de 1 kohm à porta de saída 9 do Arduino.
e) Insira os conectores do motor na protoboard, conectando um dos terminais ao pino 2 (Coletor) do transistor.
f) Conecte o outro terminal do motor à linha de alimentação positiva (vermelha) da protoboard.
g) Insira o diodo na protoboard, conectando o cátodo (terminal identificado por uma faixa no corpo do componente) à linha de alimentação positiva (vermelha) da protoboard.
h) Conecte o ânodo do diodo ao outro terminal do motor, o qual já foi conectado ao pino 2 (Coletor) do transistor.
i) Insira o potenciômetro na protoboard e conecte um dos pinos das extremidades à linha de alimentação positiva (vermelho) da protoboard.
j) Conecte o outro terminal da extremidade do potenciômetro à linha de alimentação negativa (preta ou azul) da protoboard.
k) Conecte o terminal central da protoboard à entrada analógica 0 (A0) do Arduino.
l) Conecte o polo negativo (-) do suporte para as pilhas à linha de alimentação negativa (preto ou azul) da protoboard.
m) Conecte o polo positivo (+) do suporte para as pilhas à linha de alimentação positiva (vermelha) da protoboard.



Programa

Após montar o circuito e verificar as conexões realizadas, digite o código-fonte a seguir no ambiente de desenvolvimento do Arduino.

```
// Controle de um motor de corrente contínua
// pelo potenciômetro
```

```
int MOTOR = 9;
int POT = A0;
int velocidade;

void setup() {
  pinMode(MOTOR, OUTPUT);
  pinMode(POT, INPUT);
}

void loop() {
  velocidade = analogRead(POT) / 4;
  analogWrite(MOTOR, velocidade);
  delay(50);
}
```

O conceito básico deste projeto é que, conforme o usuário girar o potenciômetro, a velocidade de rotação do motor será alterada. Para isso, o conectamos a uma das entradas analógicas do Arduino e, com a função analogRead, obtemos o valor do potenciômetro, que irá variar de zero a 1.023. Em seguida, devemos dividir esse valor por quatro, pois enquanto as entradas analógicas do Arduino trabalham com valores entre 0 e 1.023, as saídas PWM operam com valores entre 0 e 255.

10.2 Drivers e shields para motores

Com o intuito de facilitar a montagem dos projetos que utilizam motores, os circuitos de proteção comumente são implementados em placas que serão chamadas de shields ou drivers (Figura 10.6). Essas placas permitem isolar o circuito do motor dos demais do projeto e também do próprio Arduino, evitando, dessa maneira, danos.

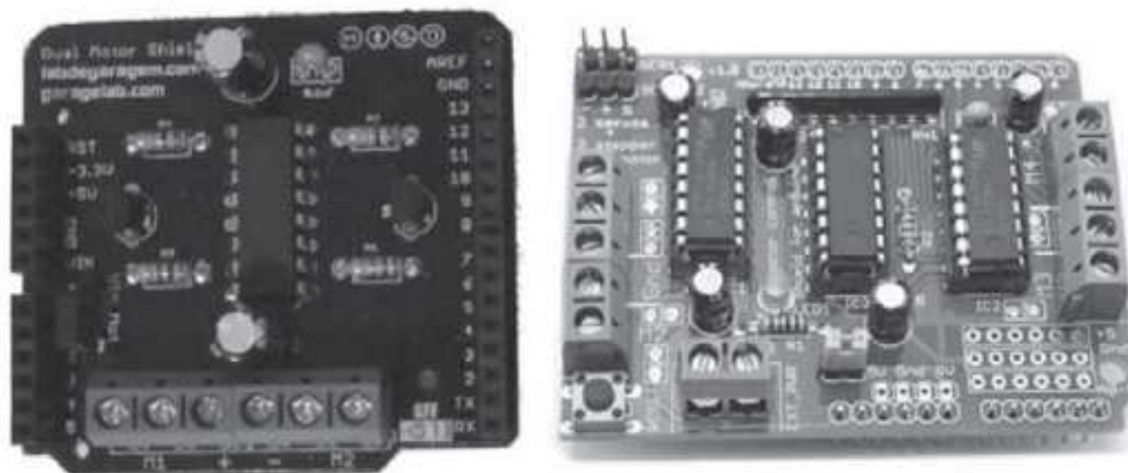


Figura 10.6 – Modelos dos módulos de motor.

Nos projetos que desenvolveremos a seguir também vamos utilizar um chassis, similar ao mostrado na Figura 10.7, que possui dois motores de corrente contínua, fato este que permite o movimento individual de cada uma das rodas.



Figura 10.7 – Chassis com dois motores de corrente contínua.



PROJETO Nº 34
Plataforma robótica com movimento autônomo

O objetivo deste projeto é utilizar sensores de ultrassom para guiarem o movimento de uma plataforma robótica. Ou seja, eles identificam obstáculos e, com base nessas informações, o sketch controla o movimento dos motores de modo a permitir que a plataforma robótica se desvie desses obstáculos (ou os evite).



Material necessário

- 1 Arduino;
- 1 shield de motor;
- 1 chassis com dois motores de corrente contínua;
- 3 sensores de ultrassom HC-SR04;
- 1 protoboard;
- 1 suporte para 4 pilhas (6,0 V);
- 1 conector para a bateria de 9 V;
- jumper cable.



Montagem do circuito

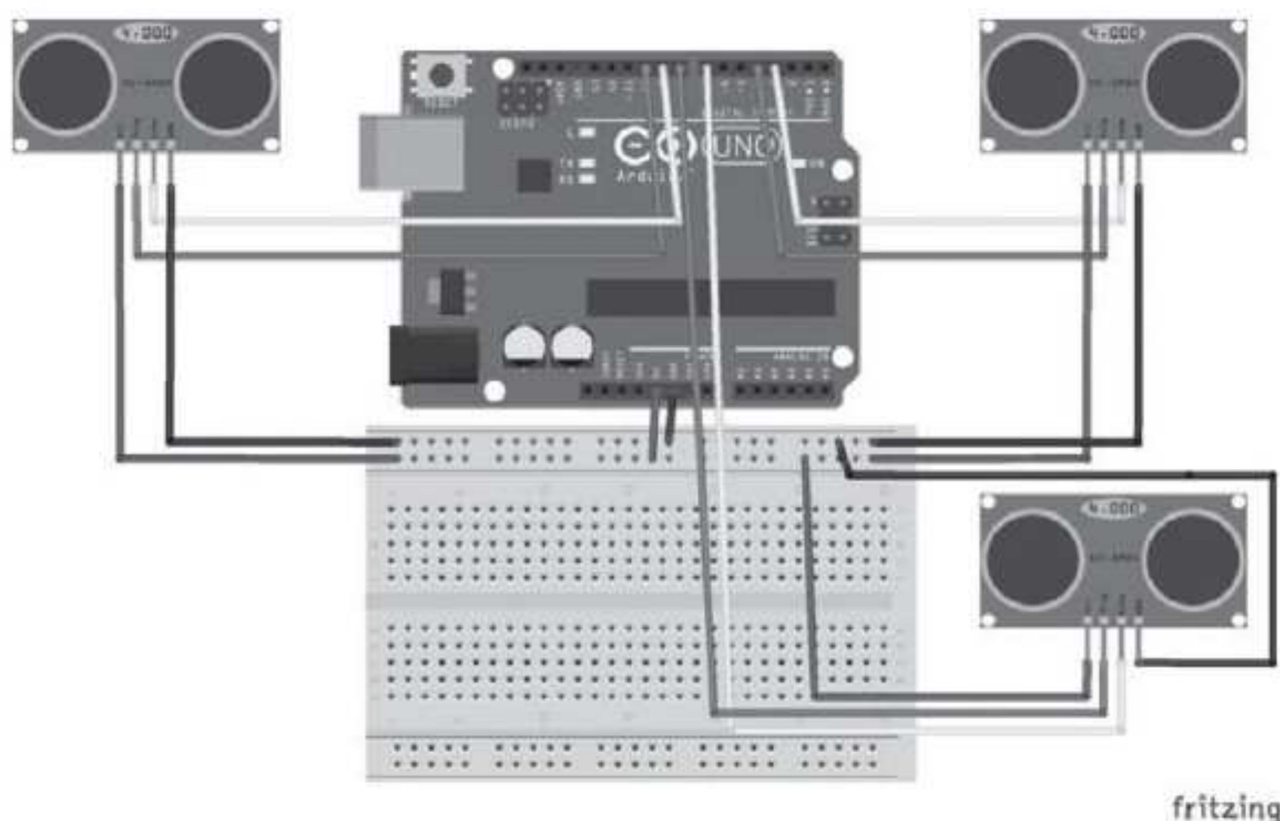


Figura 10.8 – Plataforma robótica com movimento autônomo.

Considerando como referência a Figura 10.8, realize a sequência de montagem:

a) Insira o shield de motor sobre a placa do Arduino.

b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.

c) Conecte o pino de 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.

d) Conecte o pino GND do HC-SR04 situado no lado esquerdo do chassis ligado ao GND do Arduino.

e) Conecte o pino VCC do HC-SR04 situado no lado esquerdo do chassis ligado ao 5 V do Arduino.

f) Conecte o pino ECHO do HC situado no lado esquerdo do chassis ligado ao pino 3 do Arduino.

g) Conecte o pino TRIG do HC-SR04 situado no lado esquerdo do chassis ligado ao pino 4 do Arduino.

h) Conecte o pino GND do HC-SR04 que ficará na frente do chassis ligado ao GND do Arduino.

i) Conecte o pino VCC do HC-SR04 que ficará na frente do chassis ligado ao 5 V do Arduino.

j) Conecte o pino ECHO do HC-SR04 que ficará na frente do chassis ligado ao pino 7 do Arduino.

k) Conecte o pino TRIG do HC-SR04 que ficará na frente do chassis ligado ao pino 8 do Arduino.

l) Conecte o pino GND do HC-SR04 situado no lado direito do chassis ligado ao GND do Arduino.

m) Conecte o pino VCC do HC-SR04 situado no lado direito do chassis ligado ao 5 V do Arduino.

n) Conecte o pino ECHO do HC-situado no lado direito do chassis ligado ao pino 9 do Arduino.

o) Conecte o pino TRIG do HC-SR04 situado no lado direito do chassis ligado ao pino 10 do Arduino.

Fisicamente, os sensores deverão ser colocados um na frente, outro à esquerda e o terceiro à direita da plataforma robótica, de maneira similar à apresentada na Figura 10.9.



Figura 10.9 – Posicionamento dos sensores na plataforma robótica.



Programa

O exemplo será desenvolvido utilizando o Dual Motor Shield (http://www.labdegaragem.org/loja/dual-motor-shield.html), porém pode ser facilmente adaptado para qualquer outro modelo de shield de motor. Instale a biblioteca no ambiente de desenvolvimento do Arduino e, em seguida, importe-a para o seu projeto, conforme ilustrado na Figura 10.10.

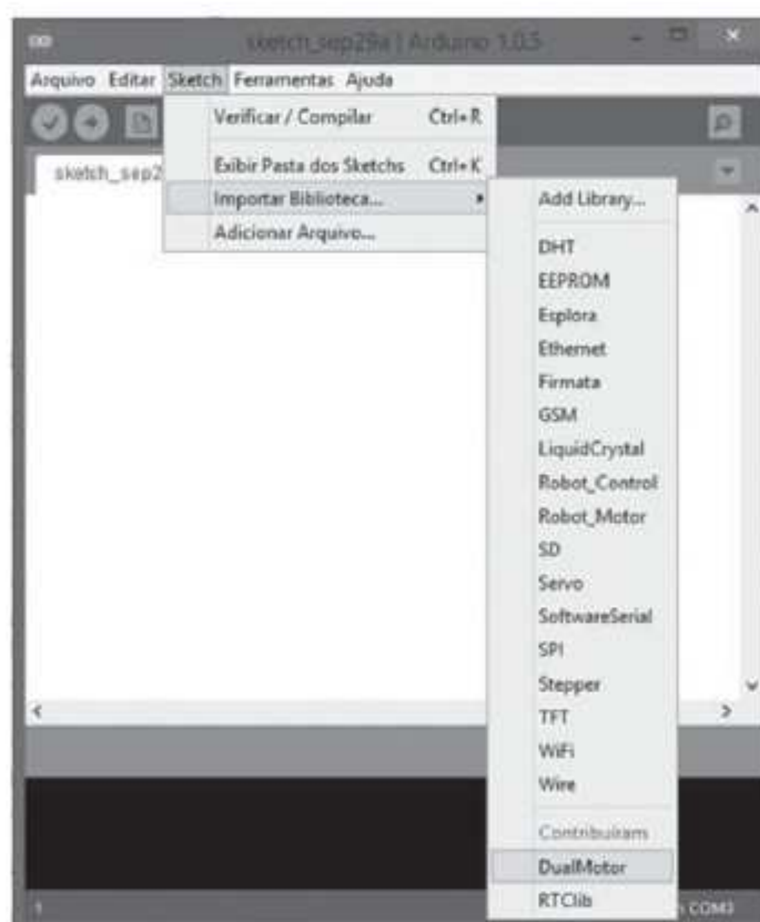


Figura 10.10 – Importação da biblioteca DualMotor.

A biblioteca apresenta um conjunto de métodos que irão definir a velocidade e o sentido do movimento de cada um dos motores (M1move e M2move) e também a parada de cada em deles (M1parar e M2parar). Dessa forma, inicialmente declaramos um objeto DualMotor, por exemplo:

```
DualMotor motores;
```

Em seguida, deverão ser usados os métodos M1Move ou M2Move. Estes recebem dois parâmetros: o primeiro serve para definir a velocidade (0 até 255) e o segundo determina o sentido de rotação do motor, ou seja, 0 para o horário ou 1 se quisermos o anti-horário. Considerando, por exemplo, o trecho de programa a seguir, estamos definindo que o motor 1 irá se mover em rotação máxima (255) no sentido horário (0).

```
motores.M1move(255,0);
```

O controle individual dos motores é fundamental para que seja possível realizar movimentos de "giro" do chassis robótico. Assim, como mostrado no trecho de programa a seguir, se paramos um motor e mantemos o outro em funcionamento, o chassis virará para a esquerda ou direita, dependendo de qual motor está parado e qual está em funcionamento.

```
motores.M1parar();
motores.M2move(255,0);
```

Após tais conceitos iniciais sobre o uso da biblioteca DualMotor, vamos agora passar para o desenvolvimento do sketch que será usado neste projeto. Dessa forma, no ambiente de desenvolvimento do Arduino, implemente o código-fonte a seguir:

```
/*
 * Biblioteca DualMotor.h é a biblioteca desenvolvida para o
 * Dual Motor Shield, pode ser gratuitamente baixada através da
 * URL http://www.labdegaragem.org/loja/dual-motor-shield.html.
 * A biblioteca é composta pelas seguintes funções:
 *
 * Instanciar um objeto:
 * DualMotor motores;
 *
 * Acionar o motor 1 velocidade (0 a 255) e sentido
 * (0 - Horário ou 1 - Anti-horário):
 * motores.M1move(velocidade, sentido);
 *
 * Acionar o motor 2 velocidade (0 a 255) e sentido
 * (0 - Horário ou 1 - Anti-horário):
 * motores.M2move(velocidade, sentido);
 *
 * Parar o motor 1:
 * motores.M1parar();
 *
 * Parar o motor 2:
 * motores.M2parar();
*/
#include <DualMotor.h>

#define VEZES 5
#define TOLERANCIA 0.90
#define TEMPO_ATUALIZACAO 200
#define E_ECHO_PIN 3
#define E_TRIG_PIN 4
#define D_ECHO_PIN 9
#define D_TRIG_PIN 10
#define F_ECHO_PIN 7
#define F_TRIG_PIN 8

DualMotor motores;
int maximo = 200; // Distância máxima: 200 cm
int minimo = 0; // Distância mínima: 0 cm
long fDuracao, eDuracao, dDuracao, fDistancia, eDistancia,
dDistancia, media;
boolean fManter;
```

```
void setup () {
  Serial.begin(9600);
  pinMode(E_TRIG_PIN, OUTPUT);
  pinMode(E_ECHO_PIN, INPUT);

  pinMode(D_TRIG_PIN, OUTPUT);
  pinMode(D_ECHO_PIN, INPUT);

  pinMode(F_TRIG_PIN, OUTPUT);
  pinMode(F_ECHO_PIN, INPUT);

  motores.M1parar(); // Parar motor1
  motores.M2parar(); // Parar motor2
}

void loop () {
  // Ler o sensor da frente
  media = 0;
  for (int i = 0; i < VEZES; i++) {
    digitalWrite(F_TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(F_TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(F_TRIG_PIN, LOW);
    fDuracao = pulseIn(F_ECHO_PIN, HIGH);
    media += fDuracao;
  }
  media = media / VEZES;
  if (fDuracao < (media * TOLERANCIA))
    fDuracao = media;
  // Calcular a distância (em cm) baseada na velocidade do som.
  fDistancia = fDuracao / 58.2;

  // Ler o sensor da esquerda
  media = 0;
  for (int i = 0; i < VEZES; i++) {
    digitalWrite(E_TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(E_TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(E_TRIG_PIN, LOW);
    eDuracao = pulseIn(E_ECHO_PIN, HIGH);
    media += eDuracao;
  }
```

```
media = media / VEZES;
if (eDuracao < (media * TOLERANCIA))
  eDuracao = media;
eDistancia = eDuracao / 58.2;

// Ler o sensor da direita
media = 0;
for (int i = 0; i < VEZES; i++) {
  digitalWrite(D_TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(D_TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(D_TRIG_PIN, LOW);
  dDuracao = pulseIn(D_ECHO_PIN, HIGH);
  media += dDuracao;
}
media = media / VEZES;
if (dDuracao < (media * TOLERANCIA))
  dDuracao = media;
dDistancia = dDuracao / 58.2;

if (fDistancia >= maximo || fDistancia <= minimo) {
  Serial.println("Frente: Fora de faixa!");
  fManter = true;
}
else {
  Serial.print("Distância frente: ");
  Serial.print(fDistancia);
  Serial.println(" cm");
  fManter = false;
}

if (eDistancia >= maximo || eDistancia <= minimo) {
  Serial.println("Esquerda: Fora de faixa!");
}
else {
  Serial.print("Distância esquerda: ");
  Serial.print(eDistancia);
  Serial.println(" cm");
}
```

```
  if (dDistancia >= maximo || dDistancia <= minimo) {
    Serial.println("Direita: Fora de faixa!");
  }
  else {
    Serial.print("Distância direita: ");
    Serial.print(dDistancia);
    Serial.println("cm");
  }
  Serial.println("");
  if (!fManter) {
    if (fDistancia < 30) {
      if (eDistancia > dDistancia) {
        motores.M1parar(); // Parar Motor1
        motores.M2move(255,0);
      }
      else {
        motores.M1move(255,0);
        motores.M2parar(); // Parar Motor2
      }
    }
    else {
      motores.M1move(255,0);
      motores.M2move(255,0);
    }
  }
  delay(TEMPO_ATUALIZACAO);
}
```

Como podemos observar neste sketch, os sensores de ultrassom obterão informações sobre os possíveis obstáculos que se encontram à frente, à esquerda e à direita do chassis. Em seguida, com base nas informações dos sensores de ultrassom, o programa decidirá se deve manter o movimento para a frente, conforme podemos notar no código-fonte a seguir.

```
if (fDistancia >= maximo || fDistancia <= minimo) {
  fManter = true;
}
else {
  fManter = false;
}
```

Depois, caso não seja possível manter o movimento para a frente, o sketch irá definir se deverá virar o chassis à esquerda ou direita, conforme podemos notar no trecho de programa a seguir.

```
if (!fManter) {
  if (fDistancia < 30) {
    if (eDistancia > dDistancia) {
      motores.M1parar(); // Parar Motor1
      motores.M2move(255,0);
    }
    else {
      motores.M1move(255,0);
      motores.M2parar(); // Parar Motor2
    }
  }
  else {
    motores.M1move(255,0);
    motores.M2move(255,0);
  }
}
```

Uma vez realizada a mudança de direção e não sendo mais detectada a presença do obstáculo que motivou a definição da nova trajetória, o chassis manterá o movimento para a frente.



PROJETO Nº 35 Plataforma robótica com controle remoto

Neste projeto vamos utilizar os conceitos de controle remoto por infravermelho, abordados anteriormente, para definir os movimentos realizados pela plataforma robótica. As teclas do controle remoto serão mapeadas de modo a produzir as próximas ações descritas.

Tecla	Ação
2	Mover para a frente
4	Mover para a esquerda
5	Parar
6	Mover para a direita
8	Mover para trás
+	Acelerar
–	Frear

Uma vez pressionada uma tecla, a ação associada irá perdurar até que uma nova tecla seja pressionada.



Material necessário

- 1 Arduino;
- 1 shield de motor;
- 1 chassis com dois motores de corrente contínua;
- 1 receptor de infravermelho;
- 1 transmissor de infravermelho (controle remoto);
- 1 LED;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

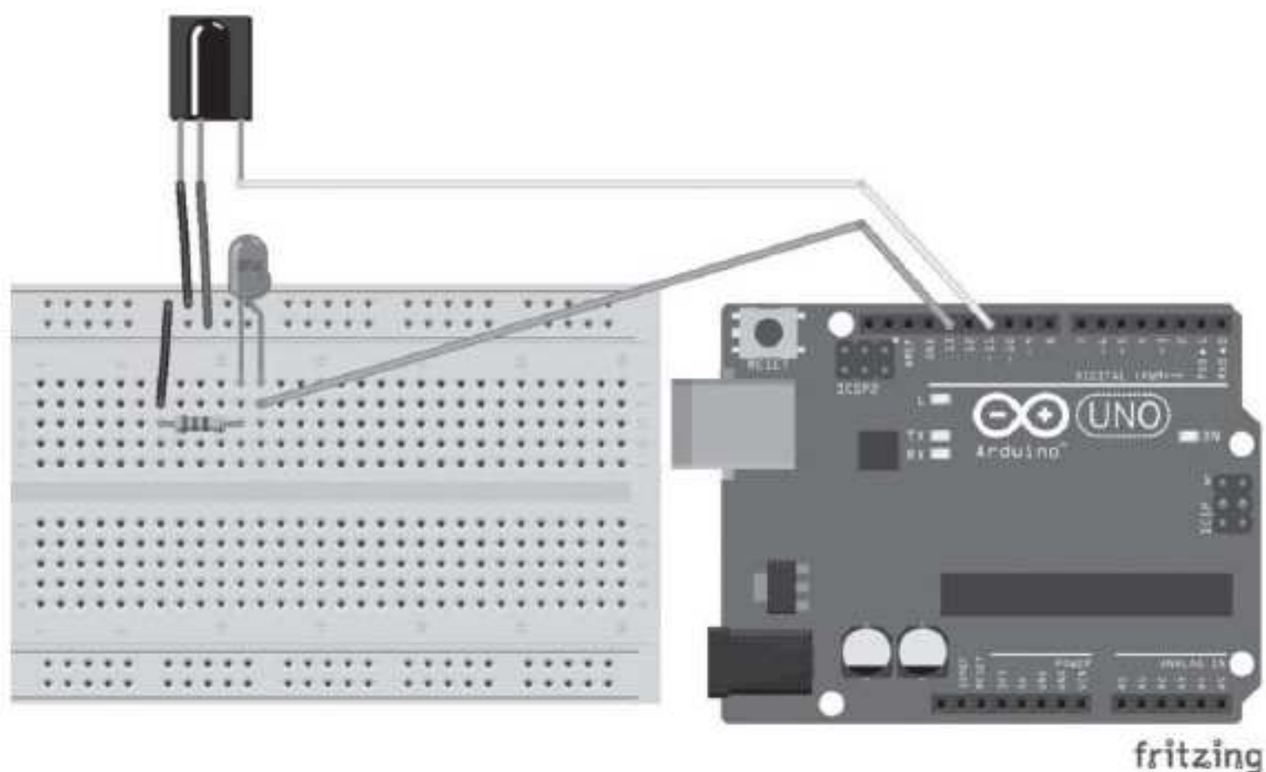


Figura 10.11 – Plataforma robótica com controle remoto.

Considerando como referência a Figura 10.11, realize a sequência de montagem:

a) Insira o shield de motor sobre a placa do Arduino.
b) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.

c) Conecte o pino de 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
d) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
e) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
f) Conecte o ânodo do LED ao pino 13 do Arduino.
g) Conecte o pino GND do receptor infravermelho na linha de alimentação GND da protoboard.
h) Conecte o pino VCC do receptor infravermelho na linha de alimentação positiva (5 V) da protoboard.
i) Conecte o pino de dados do receptor infravermelho no pino 11 do Arduino.



Programa

Copie os arquivos IRremote.h, IRremote.cpp e IRremoteInt.h da biblioteca IRemote para a pasta do seu sketch. Em seguida, digite o seguinte código-fonte. Note também que pode ser necessário que você faça o mapeamento dos valores das teclas do controle remoto em virtude dos diferentes modelos que podem ser usados.

```
#include <DualMotor.h>
#include "IRremote.h"

DualMotor motores;

int RECEPTOR = 11;
IRrecv controle(RECEPTOR);
decode_results resultado;
int velocidade = 255;

void setup() {
  // Iniciar a recepção
  controle.enableIRIn();

  // O LED conectado ao pino 13 irá piscar sempre que
  // um sinal for recebido
  controle.blink13(true);

  // Parar os motores
  motores.M1parar();
  motores.M2parar();
}
```

```
void loop() {
  if (controle.decode(&resultado)) {

    if (resultado.value == 0xFF18E7) {
      // Tecla 2 (Mover para a frente)
      motores.M1move(velocidade, 0);
      motores.M2move(velocidade, 0);
    }
    else if (resultado.value == 0xFF10EF) {
      // Tecla 4 (Mover para a esquerda)
      motores.M1move(velocidade, 0);
      motores.M2move(velocidade / 2, 0);
    }
    else if (resultado.value == 0xFF38C7) {
      // Tecla 5 (Parar)
      motores.M1parar();
      motores.M2parar();
    }
    else if (resultado.value == 0xFF5AA5) {
      // Tecla 6 (Mover para a esquerda)
      motores.M1move(velocidade / 2, 0);
      motores.M2move(velocidade, 0);
    }
    else if (resultado.value == 0xFF4AB5) {
      // Tecla 8 (Mover para trás)
      motores.M1move(velocidade, 1);
      motores.M2move(velocidade, 1);
    }
    else if (resultado.value == 0xFF906F) {
      // Tecla + (Acelerar)
      velocidade += 10;
      if (velocidade > 255)
        velocidade = 255;
    }
    else if (resultado.value == 0xFFA857) {
      // Tecla -(Frear)
      velocidade -= 10;
      if (velocidade < 0)
        velocidade = 0;
    }
```

```
    // Obter o próximo valor
    controle.resume();

    delay (50);
  }
}
```

Conforme podemos notar no trecho de programa a seguir, inicialmente declare um objeto a partir da classe DualMotor que será responsável pelo controle dos motores. Também defina o pino ao qual o receptor de infravermelho vai estar conectado, além de uma variável para obter o resultado do pressionamento de uma das teclas do controle remoto e outra que armazenará a velocidade a ser utilizada nos motores.

```
DualMotor motores;

int RECEPTOR = 11;
IRrecv controle(RECEPTOR);
decode_results resultado;
int velocidade = 255;
```

Na função setup, mostrada a seguir, devemos habilitar o receptor de infravermelho, definir que o LED conectado ao pino 13 piscará cada vez que um sinal for recebido pelo receptor e também manter os motores desligados, por meio do métodos M1parar e M2parar.

```
void setup() {
  // Iniciar a recepção
  controle.enableIRIn();

  // O LED conectado ao pino 13 irá piscar sempre que
  // um sinal for recebido
  controle.blink13(true);

  // Parar os motores
  motores.M1parar();
  motores.M2parar();
}
```

No laço principal do programa, ou seja, na função loop, devemos obter a tecla que foi pressionada no controle remoto pela função decode e, em seguida, realizar a ação solicitada. No trecho de código-fonte a seguir, mostramos a ação que será realizada quando o usuário pressionar a tecla 2 do controle remoto, isto é, mover os motores no sentido horário na velocidade definida pela respectiva variável.

```
if (controle.decode(&resultado)) {

  if (resultado.value == 0xFF6897) {
    // Tecla 2 (Mover para frente)
    motores.M1move(velocidade, 0);
    motores.M2move(velocidade, 0);
  }

  // Definição das ações das demais teclas...
}
```

Dessa maneira, cada tecla pressionada deve realizar a ação prevista. Por exemplo, para realizarmos um movimento à esquerda, devemos verificar se a tecla 4 foi pressionada, e se isso ocorreu, mantemos a velocidade do motor 1 enquanto reduzimos pela metade aquela do 2, conforme mostrado no trecho de programa a seguir.

```
else if (resultado.value == 0xFF30CF) {
  // Tecla 4 (Mover para a esquerda)
  motores.M1move(velocidade, 0);
  motores.M2move(velocidade / 2, 0);
}
```

10.3 Servomotor

Um servomotor é um motor que tem sua rotação limitada por meio de uma angulação, que pode variar entre 0° e 180° para indicar seu posicionamento. Os servomotores são muito utilizados em modelos de controle remoto, como em carrinhos, para girar o eixo de direção, ou em barcos, para direcionamento do leme. Outras aplicações interessantes são no direcionamento automatizado de antenas e na articulação de braços robóticos.



PROJETO Nº 36
Controle de um servomotor por meio de potenciômetro

O objetivo do projeto é mostrar o uso da biblioteca "Servo.h", disponível por padrão no ambiente de desenvolvimento do Arduino, para a manipulação de servomotores. Tal biblioteca apresenta um conjunto de métodos, tais como o attach, para definir qual pino está conectado e o método write que permite enviar o valor do ângulo no qual o eixo do motor deve ser posicionado.



Material necessário

- 1 Arduino;
- 1 servomotor RC padrão;
- 1 potenciômetro de 10 k;
- 1 protoboard;
- jumper cable.



Montagem do circuito

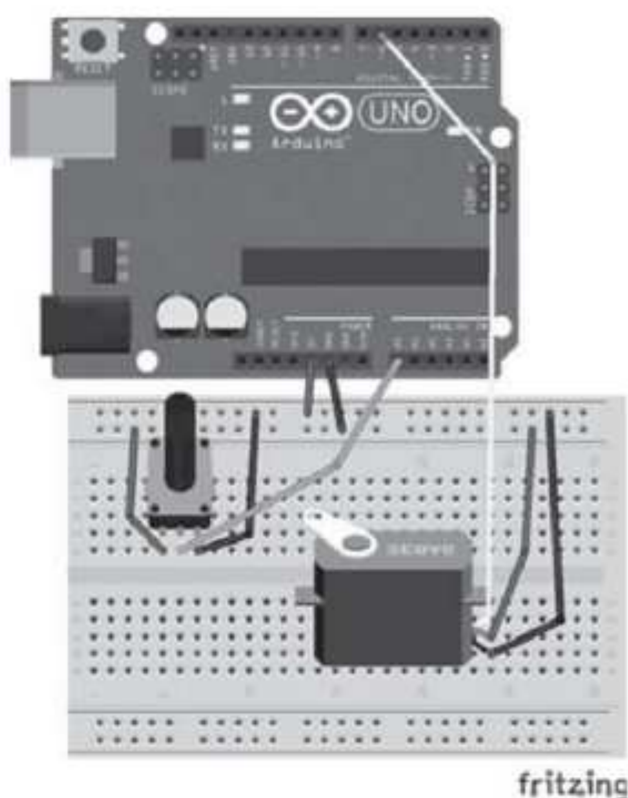


Figura 10.12 – Controle de servomotor.

Considerando como referência a Figura 10.12, realize a sequência de montagem:

a) Cabo de alimentação GND do servomotor (cabo preto) na alimentação GND da protoboard.
b) Cabo de alimentação de 5 V do servomotor (vermelho ou outra cor, logo ao lado do preto) na alimentação de 5 V da protoboard.
c) Cabo de sinal do servomotor (amarelo) no pino digital 6 do Arduino.
d) Pino positivo do potenciômetro na alimentação de 5 V da protoboard.
e) Pino negativo do potenciômetro na alimentação GND da protoboard.
f) Pino do meio (sinal) do potenciômetro no pino analógico A0 do Arduino.
g) Pino 5 V do Arduino na linha positiva da protoboard.
h) Pino GND do Arduino na linha negativa da protoboard.



Programa

No ambiente de desenvolvimento do Arduino, implemente o sketch a seguir:

```
#include "Servo.h"

Servo servo;

int POT = 0;
int SERVO = 6;
int valorPot;
int valorMotor = 0;

void setup(){
  servo.attach(SERVO);
  Serial.begin(9600);
}

void loop(){
  valorPot = analogRead(POT);
  valorMotor = map(valorPot, 0, 1023, 0, 180);
  servo.write (valorMotor);
  Serial.print(valorMotor);
  delay(20);
}
```

Neste programa utilizaremos o potenciômetro para variar a angulação do servomotor. Quanto maior o valor proveniente do sinal do potenciômetro, maior o ângulo, e vice-versa. Como a variação do potenciômetro é de 0 a 1.023 e a do servomotor é de 0 a 180, utilizamos a função map(), com o objetivo de converter o range (intervalo de variação) do potenciômetro com o do servomotor. A função map é definida como:

```
map(valor, deMin, deMax, paraMin, paraBaixo)
```

em que:

- **valor:** valor a ser convertido;
- **deMin:** valor mínimo do intervalo do valor;
- **deMax:** valor máximo do intervalo do valor;
- **paraMin:** valor mínimo do intervalo a ser convertido;
- **paraMax:** valor máximo do intervalo a ser convertido.

Basicamente, a função map() realiza uma "regra de 3" para determinar o valor de um intervalo em outro. Em nosso programa, usamos map(POT, 0, 1.023, 0, 180), e podemos entender como "o valor do potenciômetro que está no intervalo de 0 a 1.023 será em convertido em um valor equivalente no intervalo de 0 a 180".



Programa

Mantenha a mesma montagem, apenas despreze o potenciômetro, já que não será usado. Neste programa, a letra 'D' ou 'd' é digitada no Serial Monitor para fazer com que o servomotor diminua sua angulação, de 15° em 15°. Caso seja digitado 'A' ou 'a', sua angulação aumentará. No Serial Monitor, digite a letra e aperte ENTER ou o botão "Enviar" no ambiente de desenvolvimento do Arduino e insira o seguinte programa:

```
#include "Servo.h"

Servo servo;

int SERVO = 6;
char tecla;
int valorMotor=0;

void setup(){
  servo.attach(SERVO);
  Serial.begin(9600);
}

void loop(){
  tecla = Serial.read();
  if (tecla == 'D' || tecla == 'd'){
    valorMotor = valorMotor - 15;
    if (valorMotor >= 180){
      valorMotor = 180;
    }
  }
  else if (tecla == 'A' || tecla == 'a'){
    valorMotor = valorMotor + 15;
    if (valorMotor <= 0){
      valorMotor = 0;
   }
  }
  servo.write (valorMotor);
  delay(20);
}
```

Exercícios

1. Utilize um servomotor para simular um termômetro analógico, no qual uma temperatura de 0 °C lida por um sensor de temperatura LM 35 (ou DHT 11) deverá representar 0 graus de angulação do servomotor e 50 °C indicará a angulação máxima, ou seja, 180 graus. Qualquer outro valor de temperatura deverá produzir uma angulação proporcional do servomotor.

2. Desenvolva um sistema de controle para uma esteira ergométrica empregando, para isso, um motor de corrente contínua. Utilize dois botões, um para aumentar a velocidade e outro para diminuir, sempre respeitando o limite entre 0 a 255, como mostrado no Projeto Nº 33. O sistema deverá apresentar um terceiro botão de "Parada" que, quando pressionado, deverá parar a rotação do motor, ou seja, atribuir o valor 0.

11

Processing

A Processing é uma linguagem de programação de código aberto voltada a aplicações gráficas e também para as artes eletrônicas. É uma linguagem com sintaxe e estrutura bastante simples e voltada para o ensino de noções básicas de programação em um contexto gráfico. Neste capítulo utilizaremos a simplicidade e facilidade de uso da Processing para demonstrar a criação de aplicações no computador que podem se comunicar e interagir com projetos montados no Arduino.

11.1 Primeiros passos em Processing

A linguagem de programação e o próprio ambiente de desenvolvimento utilizados no Arduino foram inspirados na Processing. O ambiente de desenvolvimento da Processing pode ser baixado gratuitamente do site: http://www.processing.org/.

Antes de trabalharmos com a interação da Processing com o Arduino, vamos desenvolver alguns programas simples para nos familiarizarmos com a estrutura e os comandos da linguagem.



Programa

O primeiro passo consiste em instalar a Processing e executar o ambiente de desenvolvimento, mostrado na Figura 11.1, e que, por sinal, é muito parecido com o ambiente de desenvolvimento já utilizado no Arduino.

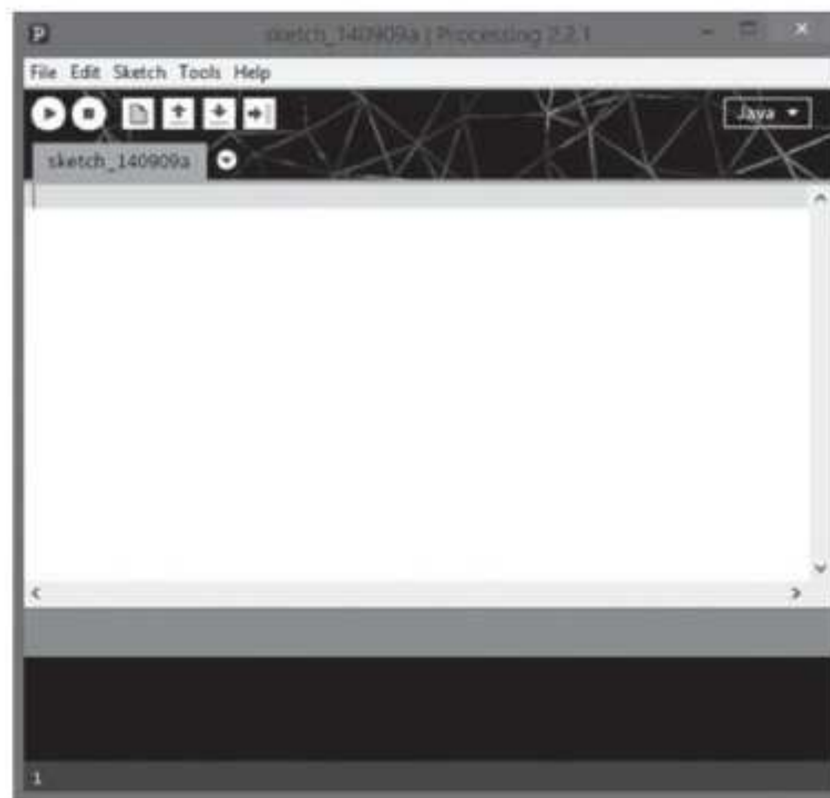


Figura 11.1 – Ambiente de desenvolvimento da Processing.

Em seguida, vamos criar um pequeno sketch para apresentar a linguagem. Para isso, simplesmente digite:

```
println("Olá Pessoal!");
```

Agora, pressione o botão Run do ambiente de desenvolvimento e observe a exibição do texto na parte inferior, conforme ilustra a Figura 11.2.



Figura 11.2 – Resultado da execução do programa.

Conforme podemos observar, a função println permite exibir as informações no console do ambiente de desenvolvimento.



Programa

Agora vamos começar a utilizar as capacidades gráficas da Processing. Para isso, digite no ambiente de desenvolvimento a seguinte função:

```
rect(10, 10, 80, 80);
```

Em seguida, pressione o botão Run do ambiente de desenvolvimento. Uma janela similar à mostrada na Figura 11.3 deverá ser exibida.



Figura 11.3 – Desenhando um retângulo.

Neste exemplo podemos observar que a função rect irá desenhar um retângulo com início nas coordenadas (10,10) e com 80 de largura e altura.



Programa

Vamos agora montar um programa completo que permitirá abordar novos conceitos da linguagem Processing. Observe no código-fonte a seguir que o programa possui duas funções:

- **setup:** a função é executada, apenas uma vez, no momento em que o programa é carregado;
- **draw:** essa função será repetidamente executada até que o programa seja encerrado.

```
void setup() {
  // Tamanho da janela
  size(300, 200);
```

```
  // Cor que será usada (0 = Preto)
  fill(0);

  // Fonte de letra e tamanho
  textFont(createFont("SansSerif", 18));

  // Alinhamento
  textAlign(CENTER);

  // Executar a função draw() apenas uma vez
  noLoop();
}

void draw() {
  // Exibir o texto no centro da janela
  text("Olá Pessoal!", width/2, height/2);
}
```

Observe os comentários que estão no código-fonte para entender cada uma das funções utilizadas. Em seguida, execute o programa, e o resultado deverá ser similar ao exibido pela Figura 11.4.



Figura 11.4 – Exibição de um texto na janela.



Programa

Uma característica marcante da Processing é a facilidade com que podemos acessar e utilizar recursos do computador. No exemplo a seguir vamos explorar a utilização do mouse dentro de um sketch:

```
void setup() {
  size (360, 180);
  fill (255); // Cor branca
}
```

```
void draw() {
  if (mousePressed) {
    ellipse(mouseX, mouseY, 100, 100);
  }
  else {
    rect(mouseX, mouseY, 100, 100);
  }
}
```

As variáveis do sistema, ou seja, previamente definidas, mousePressed, mouseX e mouseY, serão utilizadas para realizar o desenho de retângulos e elipses na janela do nosso programa. A variável mousePressed apresentará o valor lógico true (verdadeiro) quando o usuário pressionar um dos botões do mouse, caso contrário conterá false (falso). Ao executarmos o programa, é possível notar que, enquanto movemos o mouse, os retângulos são desenhados na tela, pois mousePressed estará com o valor falso e, dessa forma, os comandos que estão no else serão executados. Ao pressionar o mouse, observe que a elipse será desenhada. Na Figura 11.5 podemos visualizar a execução do programa após seguidas movimentações e cliques do mouse.



Figura 11.5 – Desenhando com o mouse.

11.2 Envio de dados do Arduino para a Processing

Após estes conceitos básicos, vamos elaborar um projeto que permita integrar um projeto desenvolvido no Arduino a um programa escrito na linguagem de programação Processing.



PROJETO Nº 37
Recebendo dados do Arduino

Neste projeto, os dados, obtidos a partir de um sensor de temperatura, são enviados pelo Arduino por meio da porta serial e recebidos por um programa desenvolvido em Processing.



Material necessário

- 1 Arduino;
- 1 LM 35 ou similar;
- 1 protoboard;
- jumper cable.



Montagem do circuito

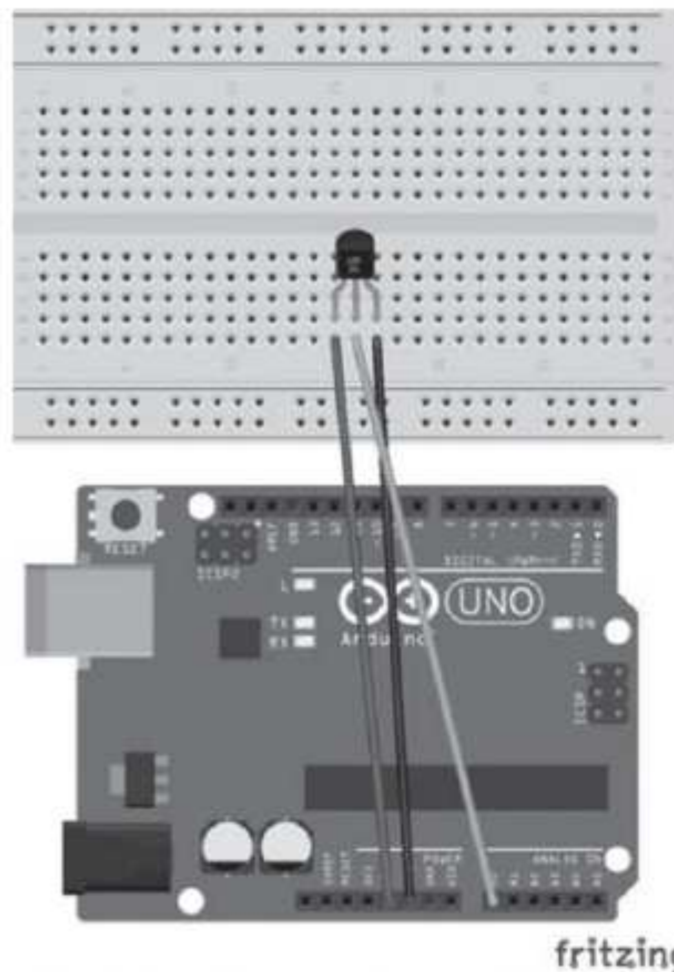


Figura 11.6 – Projeto com sensor de temperatura.

Conforme a Figura 11.6, monte o circuito da seguinte maneira:

a) Conecte o pino 1 do LM 35 à linha de alimentação positiva (5 V) da protoboard.
b) Conecte o pino 2 do LM 35 ao pino da entrada analógica A0 do Arduino.
c) Conecte o pino 3 do LM 35 à linha de alimentação negativa (GND) da protoboard.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o sketch a seguir:

```
// Pino PWM ao qual vai ser ligado o pino 2 do LM 35
const int LM 35 = A0;

// Tempo de atualização entre as leituras em ms
const int ATRASO = 5000;

// Base de conversão para Graus Celsius ((5/1023) * 100)
const float BASE_CELSIUS = 0.4887585532746823069403714565;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(lerTemperatura(), 1);
  delay(ATRASO);
}

float lerTemperatura() {
  return (analogRead(LM35) * BASE_CELSIUS);
}
```

Como podemos observar na função Serial.print utilizada neste programa, o Arduino enviará a temperatura obtida formatada com uma casa decimal.



Programa

No ambiente de desenvolvimento da Processing, digite o sketch a seguir:

```
import processing.serial.*;

Serial porta;

void setup() {
  // Exibir na saída padrão as portas disponíveis
  println(Serial.list());
```

```
  // Abrir a porta utilizada pelo Arduino
  porta = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  // Verificar se há bytes a serem lidos
  if (porta.available() > 0) {
    // Ler a String recebida
    String temperatura = porta.readStringUntil('\n');

    // Exibir na saída padrão o dado recebido
    println(temperatura);
  }
}
```

Observe nesse código-fonte que inicialmente devemos importar o pacote para comunicação serial pela diretiva import. Em seguida, na função setup, deverá ser aberta a porta serial disponível para comunicação. Note que é importante que a velocidade de comunicação seja igual nos dois programas, ou seja, tanto no sketch do Arduino quanto em Processing.

A função draw irá receber pela função readStringUntil os dados enviados pelo Arduino. Conforme podemos observar, essa função obtém os dados da entrada serial até que um determinado caractere seja recebido. No nosso exemplo, esse caractere é o '\n', ou seja, a quebra (ou fim) de linha. Em seguida, a função println exibe o dado no console do ambiente de desenvolvimento da Processing, conforme ilustrado na Figura 11.7.



Figura 11.7 – Recebendo os dados do Arduino na Processing.



Programa

Neste próximo exemplo vamos realizar a exibição da temperatura na janela gráfica do programa. Utilizaremos o mesmo projeto do Arduino, porém o sketch da Processing será desenvolvido da seguinte maneira:

```
import processing.serial.*;

Serial porta;

void setup() {
  // Título da janela
  frame.setTitle ("Termômetro Digital");

  // Tamanho da janela
  size(300, 200);

  // Cor que será usada (0 = Preto)
  fill(0);

  // Fonte de letra e tamanho
  textFont(createFont("SansSerif", 18));

  // Alinhamento
  textAlign(CENTER);

  // Exibir na saída padrão as portas disponíveis
  println(Serial.list());

  // Abrir a porta utilizada pelo Arduino
  porta = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  // Verificar se há bytes a serem lidos
  if (porta.available () > 0) {
    // Limpa a janela
    background (200);

    // Ler a String recebida
    String temperatura = porta.readStringUntil('\n');
```

```
    // Exibir o dado recebido
    text("Temperatura: " + temperatura + "°C", width/2, height/2);
  }
}
```

Na função setup iremos definir o tamanho da janela, as propriedades do texto que será exibido e abriremos a porta de comunicação serial. Na função draw, sempre que existirem dados a serem lidos na porta serial, iremos apagar o conteúdo que estava anteriormente na janela, ler o dado de temperatura recebido e, por último, exibi-lo na janela. O resultado da execução deste sketch deverá ser similar ao mostrado na Figura 11.8.



Figura 11.8 – Exibição da temperatura em um sketch da Processing.



Programa

Também podemos utilizar as facilidades da Processing em relação à programação para contexto gráfico a fim de criar um gráfico de linhas que demonstrará a variação da temperatura ao longo do tempo.

```
import processing.serial.*;

Serial porta;
int posicaoX = 10; // Posição horizontal

// Variáveis para que seja possível desenhar uma linha contínua
int ultimaPosicaoX = 0;
int ultimaAltura;

void setup () {
  frame.setTitle("Registro da Temperatura");
  size(600, 400);
  ultimaAltura = height;
  println(Serial.list());
  porta = new Serial(this, Serial.list()[0], 9600);
```

```
  background(0);
  textFont(createFont("SansSerif", 18));
  textAlign(CENTER);
  text ("Temperaturas em °C", width / 2, 24);
}

void draw () {
  if (porta.available() > 0) {
    String strTemperatura = porta.readStringUntil('\n');
    if (strTemperatura != null) {
      strTemperatura = trim(strTemperatura);
      float temperatura = float(strTemperatura);

      // Ajusta para a altura da janela
      float altura = map((temperatura * 10), 0, 1023, 0, height);

      // Desenhar uma linha entre a posição anterior e a atual
      stroke(127, 34, 255);   // Definir a cor e a
      strokeWeight(4);        // espessura da linha
      line(ultimaPosicaoX, ultimaAltura, posicaoX,
height - altura);

      // Exibir o valor da temperatura acima da linha plotada
      textFont(createFont("SansSerif", 10));
      text(strTemperatura, posicaoX, int(height - altura - 5));

      ultimaPosicaoX = posicaoX;
      ultimaAltura = int(height - altura);

      // Quando ultrapassar o comprimento da janela, o gráfico
      // deve voltar ao início
      if (posicaoX > width) {
        posicaoX = 10;
        ultimaPosicaoX = 0;
        background(0);  // Limpar a janela
        textFont(createFont("SansSerif", 18));
        text ("Registro da Temperatura", width / 2, 24);
      }
      else {
        // Incrementar a posição horizontal
        posicaoX += 30;
      }
    }
  }
}
```

A ideia básica de tal aplicação é utilizar a função line para plotar linhas entre a medida atual e a anterior da temperatura, enviada pelo Arduino e recebida pela porta serial. Na Figura 11.9 temos um exemplo de como a aplicação deverá exibir o gráfico de linha.

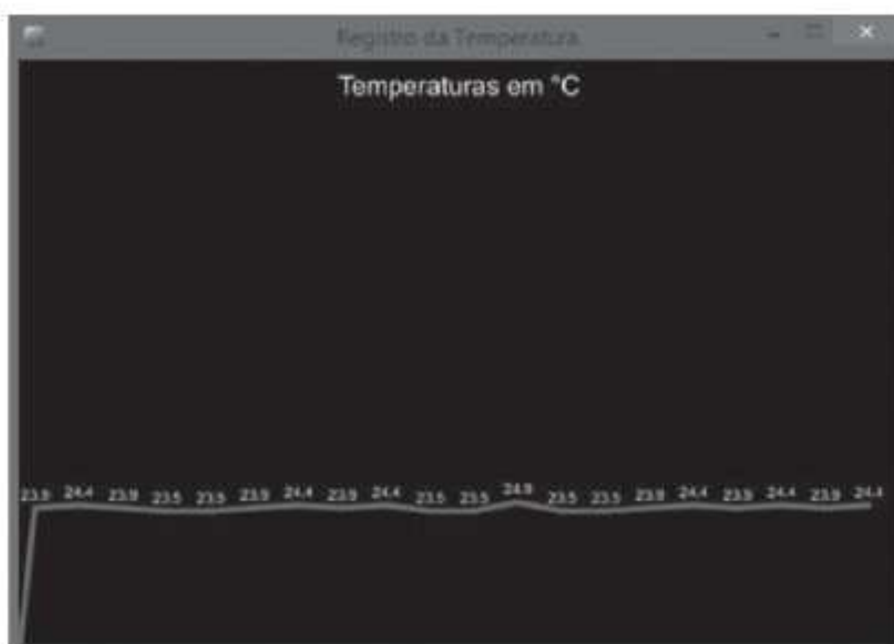


Figura 11.9 – Gráfico de linha com os valores de temperatura recebidos.

11.3 Envio de dados da Processing para o Arduino

Do mesmo modo que é possível receber dados pela porta serial e utilizá-los na aplicação desenvolvida em Processing, também podemos realizar o inverso, ou seja, fazer com que um programa escrito utilizando Processing faça o envio de dados para o Arduino.



PROJETO Nº 38
Controle de um LED pela Processing

Neste projeto utilizaremos uma aplicação gráfica, escrita em Processing, para acender ou apagar um LED.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED;
- 1 protoboard;
- jumper cable.



Montagem do circuito

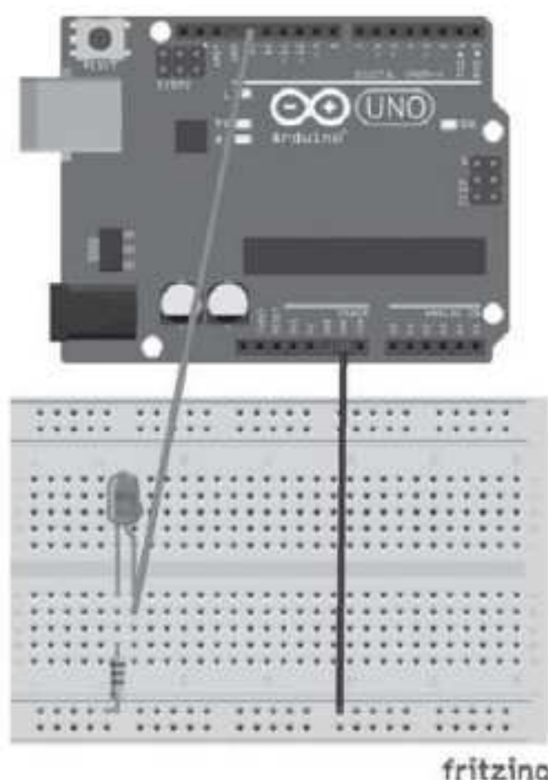


Figura 11.10 – Controle de um LED pela porta serial.

Conforme ilustra a Figura 11.10:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
c) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
d) Conecte o ânodo do LED ao pino 13 do Arduino.



Programa

Inicie o ambiente de desenvolvimento do Arduino e digite o sketch a seguir:

```
// Exemplo de entrada de dados serial

int LED = 13;
int entrada = 0;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}
```

```
void loop() {
  if (Serial.available() > 0) {
    // Ler o byte que está na entrada serial
    entrada = Serial.read();

    // Acender ou apagar o LED de acordo com o
    // valor recebido na entrada serial
    if (entrada == '0')
      digitalWrite(LED, LOW);
    else if (entrada == '1')
      digitalWrite(LED, HIGH);
  }
}
```

Conforme podemos observar neste código-fonte, o sketch acenderá ou apagará um LED conectado à porta digital 13 do Arduino. Quando o valor recebido pelo Arduino na porta serial for o caractere '0', o LED será apagado; quando ele for igual a '1', o LED será aceso.



Programa

Agora vamos desenvolver a aplicação Processing que será executada no computador e que controlará o funcionamento do LED. No ambiente de desenvolvimento da Processing, digite o sketch a seguir:

```
// Clique na imagem para mudar o estado do LED

import processing.serial.*;

Serial porta;
PImage imagem;
int valor = 0;

void setup() {
  frame.setTitle ("Controle do LED");
  size (800,480);

  println("Portas seriais disponíveis:");
  println(Serial.list());

  // Alterar para a porta disponível no computador: 1, 2, etc.
  String nomePorta = Serial.list()[0];
  porta = new Serial(this, nomePorta, 9600);
```

```
  imagem = loadImage("led-off.png");
}

void draw() {
  fill(0);
  textFont(createFont("SansSerif", 36));
  text("Controle do Led", 10, 40);

  textFont(createFont("SansSerif", 18));
  text("Clique sobre a imagem para acender ou apagar o LED.", 10, 80);

  // Exibir a imagem carregada
  image(imagem, 250, 120);
}

void mouseClicked() {
  if (valor == 0) {
    valor = 1;
    imagem = loadImage("led-on.png");
    porta.write('1');
    println("Enviando: 1");
  }
  else {
    valor = 0;
    imagem = loadImage("led-off.png");
    porta.write('0');
    println("Enviando: 0");
  }
}
```

O programa deverá enviar um valor caractere '0' ou '1' pela porta serial. Dessa forma, na função setup, inicializamos e definimos a velocidade de comunicação. Também definimos os textos que serão exibidos e as respectivas fontes e tamanhos das letras. Como podemos ver a seguir, também utilizaremos um objeto, instanciado a partir da classe PImage, que carregará, pela função loadImage, um arquivo de imagem que deverá estar presente na pasta do sketch.

```
imagem = loadImage("led-off.png");
```

A imagem carregada será posteriormente exibida juntamente com os textos criados dentro da janela da aplicação, por meio da função draw, ou seja:

```
image(imagem, 250, 120);
```

Conforme podemos observar no trecho de programa a seguir, pelo evento mouseClicked mudamos a figura e enviamos, pela porta serial, o valor '0' ou '1' que será lido pelo Arduino:

```
void mouseClicked() {
  if (valor == 0) {
    valor = 1;
    imagem = loadImage("led-on.png");
    porta.write('1');
    println("Enviando: 1");
}
  else {
    valor = 0;
    imagem = loadImage("led-off.png");
    porta.write('0');
    println("Enviando: 0");
  }
}
```

A execução do programa criado na Processing deverá produzir uma janela similar à mostrada na Figura 11.11. Quando o usuário clicar sobre a imagem, o LED na Arduino terá o seu estado modificado, ou seja, se ele estiver aceso será apagado ou vice-versa.



Figura 11.11 – Janela da aplicação "Controle do LED".

11.4 Bibliotecas

Da mesma forma que a maioria das linguagens de programação, a Processing possui inúmeras bibliotecas desenvolvidas e que podem ser utilizadas para facilitar a criação dos programas. No exemplo a seguir iremos utilizar uma biblioteca, chamada G4P, que implementa vários componentes de interface com o usuário, como, por exemplo, botões e caixas de textos, entre vários outros.

A seguir vamos instalar a biblioteca. Para isso, entre no menu Sketch, escolha a opção Import Library e, em seguida, Add Library, conforme ilustrado na Figura 11.12.

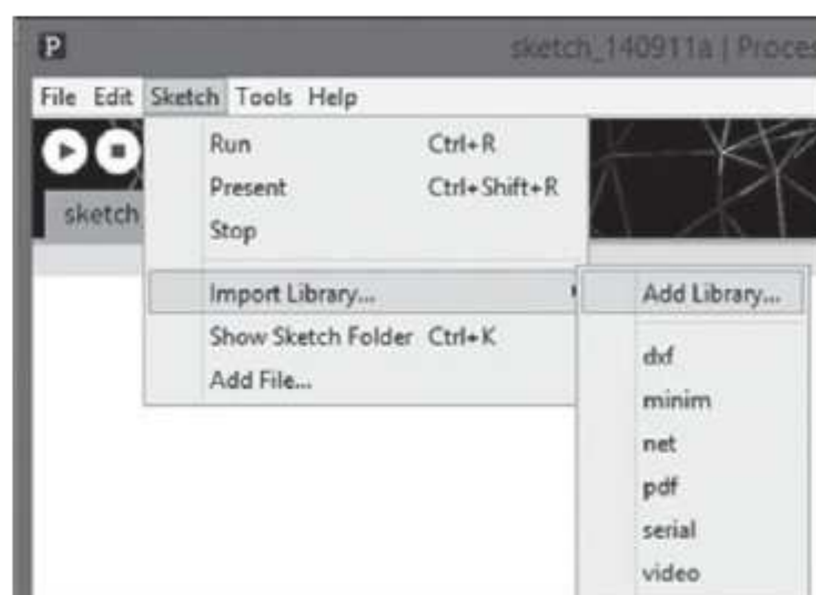


Figura 11.12 – Instalação de uma biblioteca.

Na janela Library Manager, mostrada na Figura 11.13, selecione a biblioteca desejada, no nosso caso a G4P, e clique sobre o botão Install.



Figura 11.13 – Seleção da biblioteca G4P.

Após o processo de instalação, a biblioteca estará disponível para uso e aparecerá no menu Sketch dentro da opção Import Library, conforme podemos observar na Figura 11.14.

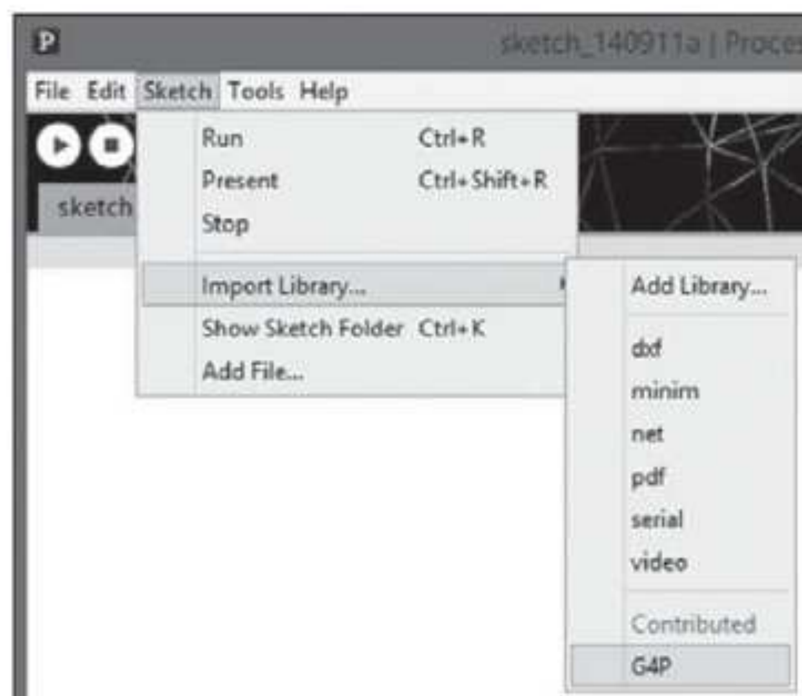


Figura 11.14 – Importar a biblioteca para o sketch.



PROJETO Nº 39
Controle da luminosidade do LED pela Processing

Neste projeto utilizaremos uma aplicação gráfica, escrita em Processing, para controlar a intensidade luminosa de um LED pelo envio de uma cadeia de caracteres (string) contendo um valor entre 0 e 255, que representam os limites dos valores da saída PWM do Arduino.



Material necessário

- 1 Arduino;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 LED (qualquer cor);
- 1 protoboard;
- jumper cable.



Montagem do circuito

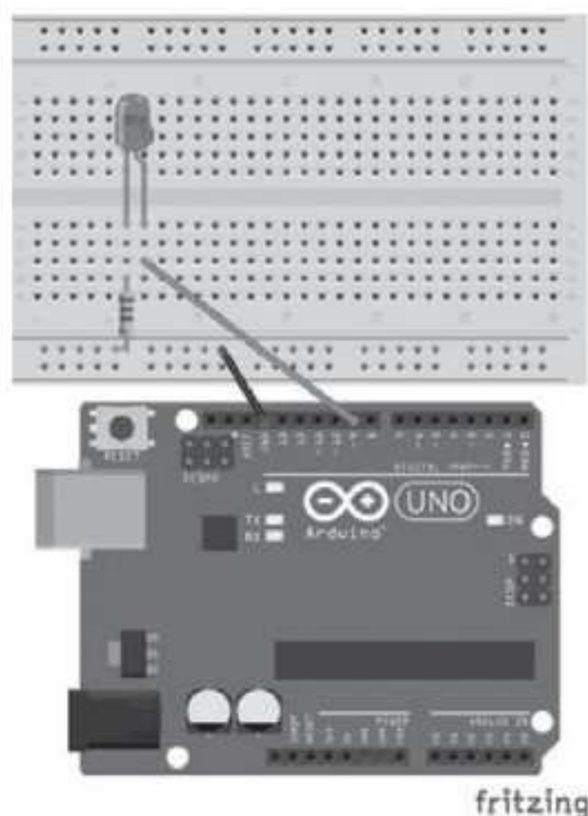


Figura 11.15 – Controle da luninosidade do LED pela porta serial.

Conforme ilustra a Figura 11.15, realize a montagem do projeto a partir dos seguintes passos:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta) da protoboard.
b) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
c) Coloque o LED com o cátodo (lado chanfrado) conectado ao resistor.
d) Conecte o ânodo do LED ao pino 9 do Arduino.



Programa

No ambiente de desenvolvimento do Arduino, digite o sketch apresentado a seguir:

```
// Exemplo de entrada de dados serial

int LED = 9;
int entrada = 0;
String strEntrada;
int valor = 0;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  strEntrada = "";
  while (Serial.available() > 0) {
    // Ler o byte que está na entrada serial
    entrada = Serial.read();
    strEntrada += (char) entrada;
  }

  if (!strEntrada.equals("")) {
    valor = strEntrada.toInt();

    // Informar o valor que foi recebido
    Serial.print("Recebido: ");
    Serial.println(valor);

    // Definir a luminosidade do LED
    analogWrite(LED, valor);
  }
  delay (500);
}
```

Analisando este programa, devemos observar que, como será necessário obter uma cadeia de caracteres a partir da entrada serial, torna-se necessário escrever um laço de repetição que irá armazenar em uma variável String todos os caracteres que foram recebidos, ou seja:

```
while (Serial.available() > 0) {
  // Ler o byte que está na entrada serial
  entrada = Serial.read();
  strEntrada += (char) entrada;
}
```

Posteriormente, a String recebida deverá ser convertida em um valor inteiro, conforme mostraremos a seguir.

```
valor = strEntrada.toInt();
```

Concluindo a análise do programa, o valor será enviado para a porta analógica do Arduino pela função analogWrite, conforme podemos observar no trecho de programa a seguir.

```
// Definir a luminosidade do LED
analogWrite(LED, valor);
```



Programa

No ambiente de desenvolvimento da Processing, vamos construir o seguinte programa, que irá fazer uso dos objetos gráficos GImageButton e GKnob, ambos disponíveis na biblioteca G4P.

```
import processing.serial.*;
import g4p_controls.*;

Serial porta;
GKnob knob;
GImageButton botao;
String[] imagens;

int valorAnalogico = 0;

void setup() {
  frame.setTitle ("Controle do LED");
  size (600, 480);

  knob = new GKnob(this, 210, 150, 180, 180, 0.8);
  knob.setTurnRange(110.0, 70.0);
  knob.setTurnMode(GKnob.CTRL_ANGULAR);
  knob.setValue(0.0);

  imagens = new String[] {"botao.png", "botao.png",
    "botao-press.png" };
  botao = new GImageButton(this, 243, 400, imagens);

  println("Portas seriais disponíveis:");
  println(Serial.list());

  // Alterar para a porta disponível no computador: 1, 2, etc.
  String nomePorta = Serial.list()[0];
  porta = new Serial(this, nomePorta, 9600);
}
```

```
void draw(){
  background(200);
  fill(0);

  textFont(createFont("SansSerif", 36));
  text("Controle da Luminosidade do LED", 10, 40);

  textFont(createFont("SansSerif", 18));
  text("Valor: " + valorAnalogico, 10, 80);
}

public void handleKnobEvents(GValueControl controle,
GEvent evento) {
  if (controle == knob) {
    valorAnalogico = int(knob.getValueF() * 255);
  }
}

public void handleButtonEvents(GImageButton controle,
GEvent evento) {
  if (controle == botao) {
    println("Enviando: " + valorAnalogico);
    porta.write(valorAnalogico + "\n");
  }
}
```

Utilizando a biblioteca G4P, vamos criar um objeto a partir da classe GKnob, que consiste em um elemento gráfico que simula o funcionamento de um potenciômetro e é bastante útil para trabalharmos com faixas de valores. Conforme podemos observar no trecho de programa a seguir, quando criamos o objeto devemos passar o contexto gráfico no qual ele será apresentado. Neste caso, usamos a palavra reservada 'this' para expressar que o objeto será criado no contexo atual. Então, os dois parâmetros seguintes definem a posição na qual o objeto será colocado, enquanto os próximos dois parâmetros definem o tamanho do objeto. O último parâmetro define a largura na qual será apresentada a escala de valores do objeto.

Após criarmos o objeto, devemos utilizar o método setTurnRange, que permitirá definir os ângulos de início e fim da rotação (giro). Em seguida, setTurnMode define a forma como ocorrerá o giro do controle. Em tal exemplo, utilizaremos a opção de controle angular. Por último, definimos o valor inicial pelo método setValue, que, no nosso caso, será zero.

```
knob = new GKnob(this, 210, 150, 180, 180, 0.8);
knob.setTurnRange(110.0, 70.0);
knob.setTurnMode(GKnob.CTRL_ANGULAR);
knob.setValue(0.0);
```

O próximo objeto que criaremos será a partir da classe GImageButton, ou seja, um botão que permite apresentar uma imagem. Como ilustrado pelo trecho de programa a seguir, inicialmente devemos declarar um vetor com as imagens. A primeira imagem é padrão, ou seja, é mostrada quando o botão não está ativo; a segunda é apresentada quando o ponteiro do mouse é colocado sobre a imagem; e a última é utilizada quando o botão recebe o clique do mouse. Em seguida, o objeto é instanciado passando como parâmetros o contexto gráfico, a sua posição na janela e o vetor com as imagens que deverão ser exibidas.

```
imagens = new String[] {"botao.png", "botao.png",
"botao-press.png" };
botao = new GImageButton(this, 243, 400, imagens);
```

Na função draw devemos definir a cor de fundo da janela, a cor do texto e exibir os textos desejados, conforme ilustrado a seguir. Também é importante observar que os objetos criados a partir da biblioteca G4P não precisam ser desenhados, pois, uma vez criados, já são parte do contexto gráfico e são atualizados automaticamente.

```
void draw(){
  background(200);
  fill(0);

  textFont(createFont("SansSerif", 36));
  text("Controle da Luminosidade do LED", 10, 40);

  textFont(createFont("SansSerif", 18));
  text("Valor: " + valorAnalogico, 10, 80);
}
```

A etapa seguinte na construção do programa consiste em realizar o tratamento dos eventos gerados pelo botão e pelo knob. No knob vamos definir o valor inteiro que será enviado para o Arduino. Como uma saída PWM trabalha com valores entre 0 e 255 e o knob gera um float entre 0,0 e 1,0, precisamos garantir que o valor do knob também esteja na mesma faixa. Para obtermos isso, basta pegar o valor atual do knob pelo método getValueF, multiplicá-lo por 255 e converter o resultado para inteiro por meio da função int, como podemos visualizar a seguir.

```
public void handleKnobEvents(GValueControl controle,
GEvent evento) {
  if (controle == knob) {
    valorAnalogico = int(knob.getValueF() * 255);
  }
}
```

Concluindo a aplicação, quando o usuário clicar no botão devemos enviar o valor do knob para a porta serial por meio do método write, ou seja:

```
public void handleButtonEvents(GImageButton controle,
GEvent evento) {
  if (controle == botao) {
    println("Enviando: " + valorAnalogico);
    porta.write(valorAnalogico + "\n");
  }
}
```

Ao executar o programa, devemos ter uma tela similar à exibida na Figura 11.16. Observe o controle knob e logo abaixo o botão com a imagem.



Figura 11.16 – Aplicação para controle da luminosidade do LED.



PROJETO Nº 40
Envio de textos para uma matriz de LEDs

Aplicando o conceito de matrizes de LEDs abordado anteriormente, vamos elaborar uma aplicação na Processing que receberá um texto, digitado pelo usuário, e o enviará por meio da porta serial para um sketch Arduino, o qual o exibirá em uma matriz de LEDs.



Material necessário

- 1 Arduino;
- 2 matrizes de LED de 8x8*;
- 2 CI MAX 7219 ou 7221*;
- 2 resistores de 100 kohms (marrom, preto, amarelo)*;
- 1 protoboard;
- jumper cable.

* Podem ser substituídos por dois módulos de matrizes de LEDs.



Montagem do circuito

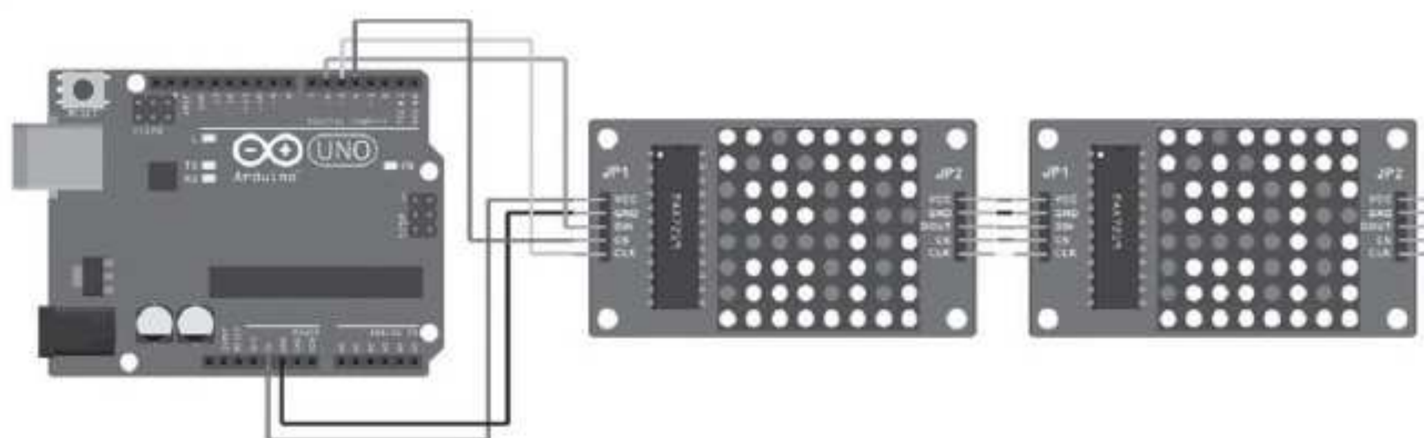


Figura 11.17 – Envio de textos para as matrizes de LEDs.

Conforme ilustrado no diagrama mostrado na Figura 11.17, realize os seguintes passos necessários para a montagem do projeto:

a) Conecte o pino VCC do módulo ao pino 5 V do Arduino.
b) Conecte o pino GND do módulo ao pino GND do Arduino.
c) Conecte o pino DIN do primeiro módulo ao pino 6 do Arduino.
d) Conecte o pino CS (LOAD/CS) do primeiro módulo ao pino 4 do Arduino.
e) Conecte o pino CLK (CLOCK) do primeiro módulo ao pino 5 do Arduino.
f) Conecte o pino DOUT (Data Out) do primeiro módulo ao pino DIN do segundo módulo.
g) Conecte o pino CS (LOAD/CS) do primeiro módulo ao pino CS do segundo módulo.
h) Conecte o pino CLK (CLOCK) do primeiro módulo ao pino CLK do segundo módulo.



Programa

Digite o sketch a seguir no ambiente de desenvolvimento do Arduino, lembrando de inserir no projeto as bibliotecas necessárias, ou seja, avr/pgmspace.h, LEDControl.h e FonteMatriz.h.

```
#include <avr/pgmspace.h>
#include "LEDControl.h"
#include "FonteMatriz.h"

const int NUM_MATRIZ = 2;

/*
 * Criar um LEDControl (matrizLed).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino (NUM_MATRIZ)
 */
LEDControl lc=LEDControl(6, 5, 4, NUM_MATRIZ);

String linha = "";
char comando = 'R';
String mensagem = "Envie um texto.";
int atraso = 500;

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < NUM_MATRIZ; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 2);
    lc.clearDisplay(i);
  }
}
```

```
boolean obterComando() {
  linha = "";
  while (Serial.available() > 0) {
    // Ler o byte que está na entrada serial
    linha += (char) Serial.read();
  }
  if (!linha.equals("")) {
    comando = linha.charAt(0);

    if (comando == 'R') {
      if (linha.substring(1, linha.length()).toInt() > 0)
        atraso = linha.substring(1, linha.length()).toInt();
    }
    else if (comando == 'T') {
      if (!linha.substring(1, linha.length()).equals(""))
        mensagem = linha.substring(1, linha.length());
}
return (true);
  }
  else
    return (false);
}

void loop() {
int contador = 0;
  int caractere = 0;
int caractereAnterior = 0;

  do {
    // Verificar a porta serial
    if (obterComando() && comando == 'T')
      break;

    if (comando != 'P') {
      // Obtém cada caractere que compõe a mensagem
      caractere =  mensagem.charAt(contador);
      if (caractere != 0) {
        exibirCaractere(1, caractere);
        exibirCaractere(0, caractereAnterior);
        caractereAnterior = caractere;
        delay(atraso);
      }
      contador++;
    }
  }
```

```
  while (caractere != 0);
  lc.clearDisplay(0);
}

void exibirCaractere(int matriz, int ascii) {
  if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
    for (int i = 0; i < 7; i++) {
      // Busca na tabela de caracteres uma linha
      int linha =
        pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
      lc.setColumn(matriz, 7 - i, linha);
    }
  }
}
```

Conforme podemos notar no trecho de código-fonte a seguir, neste programa receberemos uma cadeia de caracteres (String) pela porta serial. Para obter os dados, devemos montar um laço de repetição que obterá um caractere da entrada serial pelo comando read e, em seguida, acumular na variável String linha.

```
linha = "";
while (Serial.available() > 0) {
  // Ler o byte que está na entrada serial
  linha += (char) Serial.read();
}
```

Na String linha, o primeiro caractere irá indicar o comando a ser realizado, que pode ser "R" para rolar o texto com o atraso especificado em seguida. Por exemplo, R500 definirá que a rolagem de texto ocorrerá com um atraso de 500 ms para deslocamento dos caracteres. Para obter o comando, que é a primeira letra da String recebida, vamos usar o método charAt, conforme podemos observar a seguir.

```
comando = linha.charAt(0);
```

Obtemos então a duração utilizando o método substring e, em seguida, convertendo-o para um valor inteiro pelo método toInt, conforme ilustrado no comando a seguir.

```
atraso = linha.substring(1, linha.length()).toInt();
```

O comando "T" definirá que o texto a ser mostrado nas matrizes de LEDs será alterado. Por exemplo, "TBom dia!" alterará o texto exibido para "Bom dia!". Para obter o texto a ser exibido, utilizaremos o seguinte comando:

```
mensagem = linha.substring(1, linha.length());
```

Em seguida, vão sendo exibidos nas matrizes de LEDs dois caracteres por vez da String mensagem pela função exibirCaractere, conforme mostrado no seguinte trecho de programa.

```
do {
// Verificar a porta serial
  if (obterComando() && comando == 'T')
    break;

  if (comando != 'P') {
    // Obtém cada caractere que compõe a mensagem
    caractere =  mensagem.charAt(contador);
    if (caractere != 0) {
      exibirCaractere(1, caractere);
      exibirCaractere(0, caractereAnterior);
      caractereAnterior = caractere;
      delay(atraso);
    }
    contador++;
  }
}
while (caractere != 0);
```

Também é possível observar que a rolagem apenas irá ocorrer se o comando enviado for diferente de 'P', ou seja, pausar.



Programa

No ambiente de desenvolvimento da Processing, digite o seguinte sketch:

```
import processing.serial.*;
import g4p_controls.*;
import java.awt.Font;
```

```
Serial porta;
GLabel titulo;
GLabel txtTexto;
GLabel txtVelocidade;
GTextField texto;
GSlider velocidade;
GImageButton enviar;
GImageButton rolar;
GImageButton pausar;
String[] imagens;

int valorAnalogico = 0;

void setup() {
  frame.setTitle ("Display de LEDs");
  size (600, 400);

  titulo = new GLabel(this, (width / 2) -  200, 10, 400, 44);
  titulo.setText("Display de LEDs");
  titulo.setFont (new Font("SansSerif", Font.BOLD, 36));
  titulo.setTextAlign(GAlign.CENTER, null);

  txtTexto = new GLabel(this, 20, 90, 400, 24);
  txtTexto.setText("Digite o texto a ser enviado para o display:");
  txtTexto.setFont (new Font("SansSerif", Font.BOLD, 18));
  txtTexto.setTextAlign(GAlign.LEFT, null);

  texto = new GTextField(this, 20, 120, 560, 32);
  texto.setFont (new Font("SansSerif", Font.PLAIN, 18));

  txtVelocidade = new GLabel(this, 20, 190, 400, 24);
  txtVelocidade.setText("Velocidade da rolagem:");
  txtVelocidade.setFont (new Font("SansSerif", Font.BOLD, 18));
  txtVelocidade.setTextAlign(GAlign.LEFT, null);

  velocidade = new GSlider(this, 20, 220, 560, 20, 15);
  velocidade.setLocalColorScheme(6);
  velocidade.setValue(0.750);

  imagens = new String[] {"enviar-texto.png", "enviar-texto.png",
"enviar-texto-press.png" };
  enviar = new GImageButton(this, 168, 290, imagens);
  enviar.tag = "Enviar o texto";
```

```
  imagens = new String[] {"play.png", "play.png",
    "play-press.png" };
  rolar = new GImageButton(this, 268, 290, imagens);
  rolar.tag = "Rolar o texto";

  imagens = new String[] {"pause.png", "pause.png",
    "pause-press.png" };
  pausar = new GImageButton(this, 368, 290, imagens);
  pausar.tag = "Pausar";

  println("Portas seriais disponíveis:");
  println(Serial.list());

  // Alterar para a porta disponível no computador: 1, 2, etc.
  String nomePorta = Serial.list()[0];
  porta = new Serial(this, nomePorta, 9600);
}

void draw() { }

public void handleButtonEvents(GImageButton controle,
GEvent evento) {
  if (controle == enviar && !texto.getText().equals("")) {
    println("Enviando: " + texto.getText());
    porta.write("T" + texto.getText() + " \n");
  }
  else if (controle == rolar) {
    int atraso = (int) (2000 - (velocidade.getValueF() * 2000));
    println("Enviando: R" + atraso);
    porta.write("R" + atraso + "\n");
  }
  else if (controle == pausar) {
    println("Enviando: P");
    porta.write("P\n");
  }
}
```

Na função setup iremos montar o layout de janela mostrado na Figura 11.18. Note que serão utilizados objetos da classeGLabel para mostrar os trechos estáticos. Um objeto instanciado pela classe GTextField irá receber o texto digitado pelo usuário. O controle da velocidade será realizado por meio do controle GSlider, e os botões serão implementados por objetos instanciados da classe GImageButton.



Figura 11.18 – Sketch em Processing para controle das matrizes de LEDs.

O método handleButtonEvents será executado sempre que o usuário pressionar um dos três botões que estão na janela.

```
public void handleButtonEvents(GImageButton controle,
GEvent evento) {
  if (controle == enviar && !texto.getText().equals("")) {
    println("Enviando: " + texto.getText());
    porta.write("T" + texto.getText() + " \n");
  }
  else if (controle == rolar) {
    int atraso = (int) (2000 - (velocidade.getValueF() * 2000));
    println("Enviando: R" + atraso);
    porta.write("R" + atraso + "\n");
  }
  else if (controle == pausar) {
    println("Enviando: P");
    porta.write("P\n");
  }
}
```

Dentro deste método, devemos identificar o botão que foi pressionado. Por exemplo, a expressão da estrutura if mostrada a seguir será verdadeira quando o botão enviar for pressionado.

```
if (controle == enviar)
```

Quando o botão enviar for pressionado, enviamos por meio da porta serial uma String contendo o comando "T" na primeira posição, seguido do texto digitado pelo usuário e

finalizado com um fim de linha (" \n"), conforme podemos observar na linha de programa a seguir.

```
porta.write("T" + texto.getText() + " \n");
```

No trecho do programa mostrado a seguir, devemos notar que quando o usuário pressionar o botão rolar, devemos obter a posição do slider pelo método getValueF, converteremos em uma escala entre 0 e 2.000, que no sketch do Arduino será o atraso na rolagem, ou seja, um valor entre 0 e 2 segundos.

```
int atraso = (int) (2000 - (velocidade.getValueF() * 2000));
println("Enviando: R" + atraso);
porta.write("R" + atraso + "\n");
```

Concluindo a análise do código-fonte, quando o botão pausar for pressionado, devemos enviar o comando "P" pela porta serial.

Exercícios

1. Elabore um sketch em Processing que desenhe um retângulo, preenchido na cor azul, nas coordenadas (10, 10) com comprimento de 50 cm e altura de 80 cm.
2. Utilizando apenas retângulos, escreva um sketch em Processing que reproduzirá a imagem mostrada na Figura 11.19.



Figura 11.19 – Imagem a ser reproduzida.

3. Elabore um sketch para desenhar a imagem mostrada na Figura 11.20.

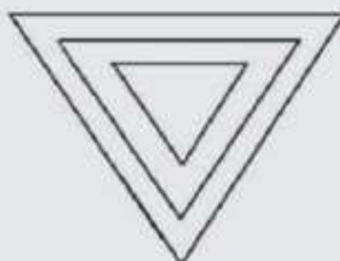


Figura 11.20 – Triângulos.

4. Desenvolva um sketch em Processing com três controles do tipo knob (Figura 11.21) para controlar a intensidade de cada uma das cores de um LED RGB:



Figura 11.21 – Controle de um LED RGB.

5. Altere o projeto display de LEDs acrescentando a ele um botão de Reset, conforme mostrado na Figura 11.22. Ao pressionar esse botão o texto a ser exibido na matriz de LEDs deverá ser alterado para "Envie um texto." e o atraso deverá ser ajustado para 500.



Figura 11.22 – Projeto com o botão de Reset.

12 Android e Arduino

Utilizando os conceitos abordados sobre conexão Bluetooth, podemos elaborar projetos que podem interagir com diversas plataformas, como Windows, Linux e Android. Neste capítulo focaremos a integração com o Android; dessa forma, um smartphone ou tablet com esse sistema operacional poderá trabalhar em conjunto com projetos elaborados para o Arduino. O objetivo não será ensinar a programação para o Android, mas apenas focar nos aspectos práticos que permitirão criar os aplicativos necessários aos nossos projetos.

12.1 Introdução e configuração do ambiente de desenvolvimento

Como qualquer outra tecnologia, devemos considerar que o desenvolvimento de projetos para o Android requer o uso de um ambiente de desenvolvimento. Para realizar o seu download, vá até a página http://developer.android.com/sdk/index.html e escolha a opção "Download Eclipse ADT with the Android SDK". Esse ambiente não precisa ser instalado, basta descompactar o conteúdo do arquivo de instalação em uma pasta.

O Eclipse pode ser executado pela pasta eclipse localizada dentro daquela na qual foram descompactados os arquivos. Após abrir o programa, identifique o ícone "Android SDK Manager" localizado na barra de tarefas do Eclipse, conforme ilustrado na Figura 12.1.

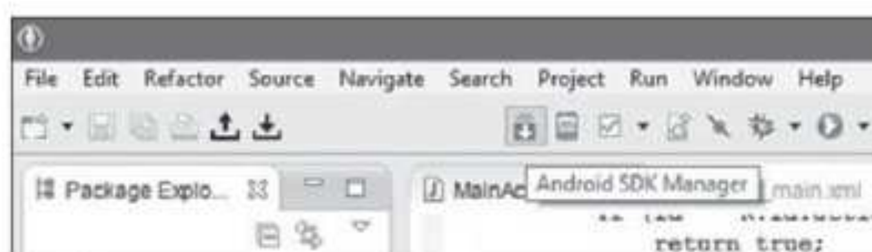


Figura 12.1 – Acesso ao Android SDK Manager a partir do Eclipse.

Na janela do "Android SDK Manager", mostrada na Figura 12.2, selecione para qual versão do Android você deseja desenvolver os aplicativos. Os exemplos deste livro funcionam em qualquer dispositivo com Android 4.0 ou superior.

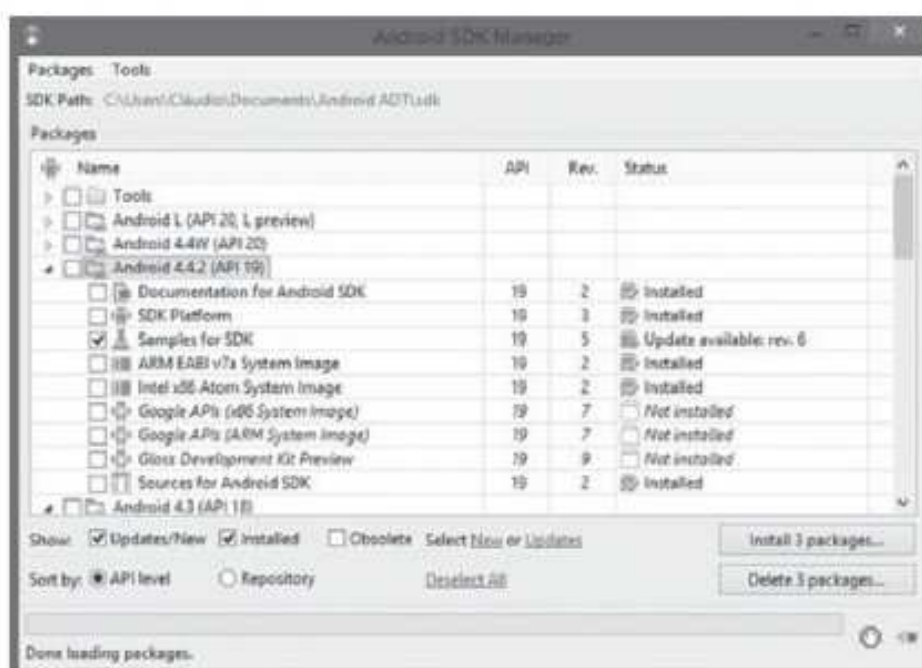


Figura 12.2 – Janela do Android SDK Manager.

Após instalar os arquivos do SDK necessários, devemos definir a máquina virtual Android a ser utilizada. Para isso, feche o "Android SDK Manager" e no Eclipse escolha o ícone do "Android Virtual Device Manager" que está na barra de tarefas, conforme podemos observar na Figura 12.3.

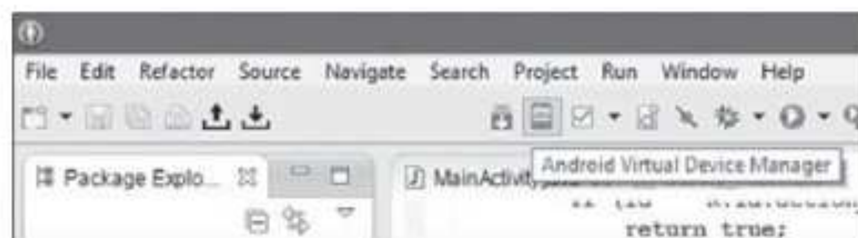


Figura 12.3 – Acesso ao Android Virtual Device Manager a partir do Eclipse.

Na janela que será mostrada (Figura 12.4), pressione o botão "Create..." para criar a máquina virtual que será usada para testarmos as aplicações.

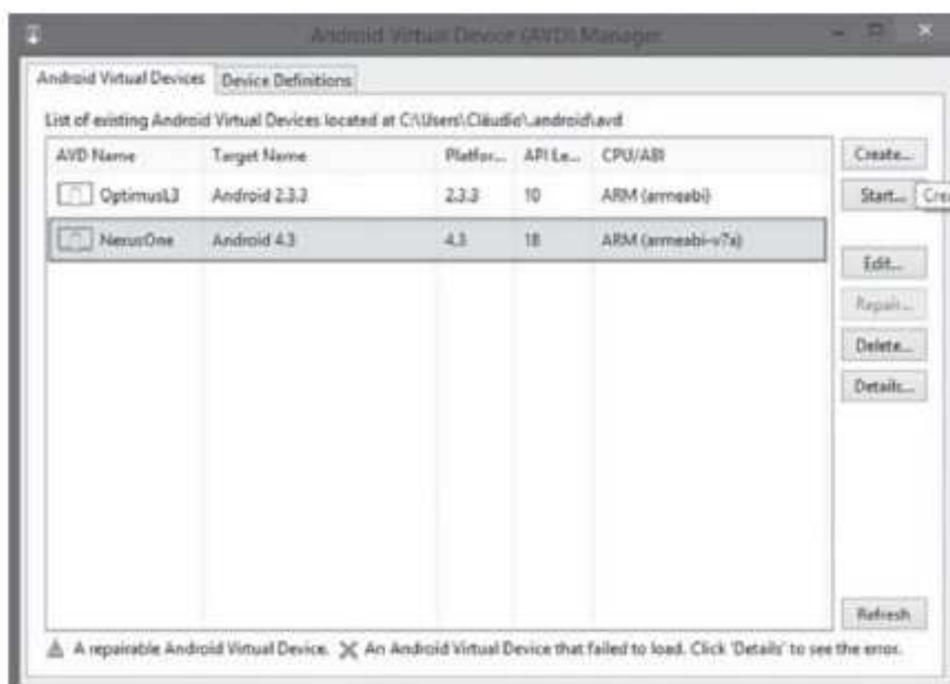


Figura 12.4 – Criação de uma máquina virtual.



Programa

Agora estamos prontos para criar uma primeira aplicação para Android. Para isso, no Eclipse, escolha a opção "Android Application Project" disponível no menu "File", "New", conforme mostrado na Figura 12.5.

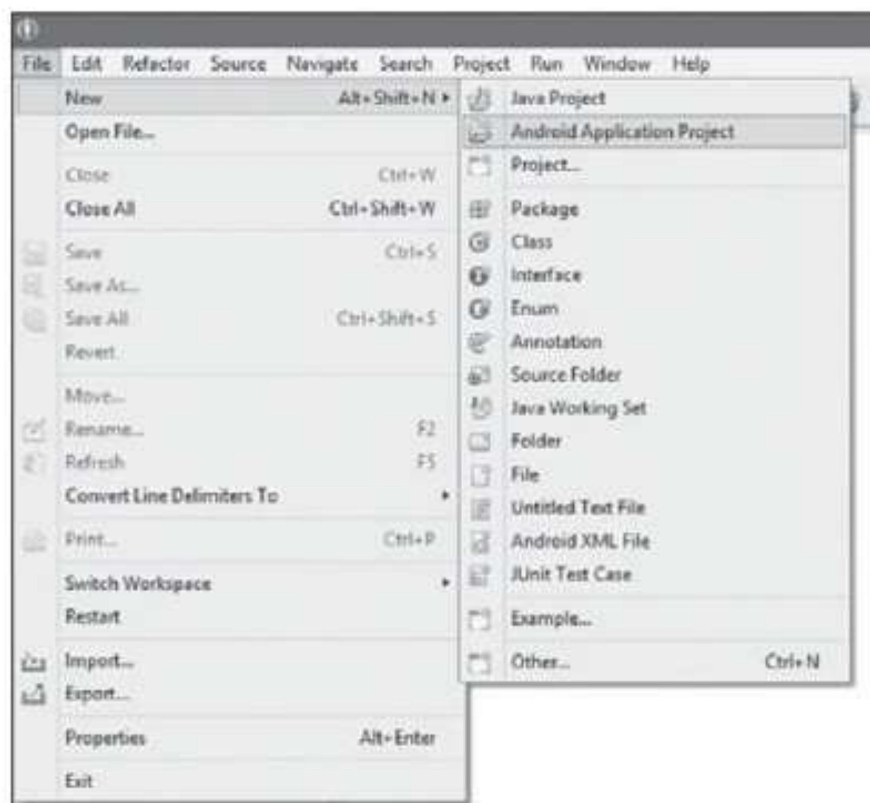


Figura 12.5 – Criar um novo projeto de aplicativo Android.

Conforme mostrado na Figura 12.6, defina o nome do projeto, forneça as informações a respeito do SDK e pressione o botão "Next>".



Figura 12.6 – Definição do nome do projeto e do SDK utilizado.

Configure o projeto, aceitando o valor padrão dos campos na janela mostrada na Figura 12.7, e pressione o botão "Next >" para prosseguir.

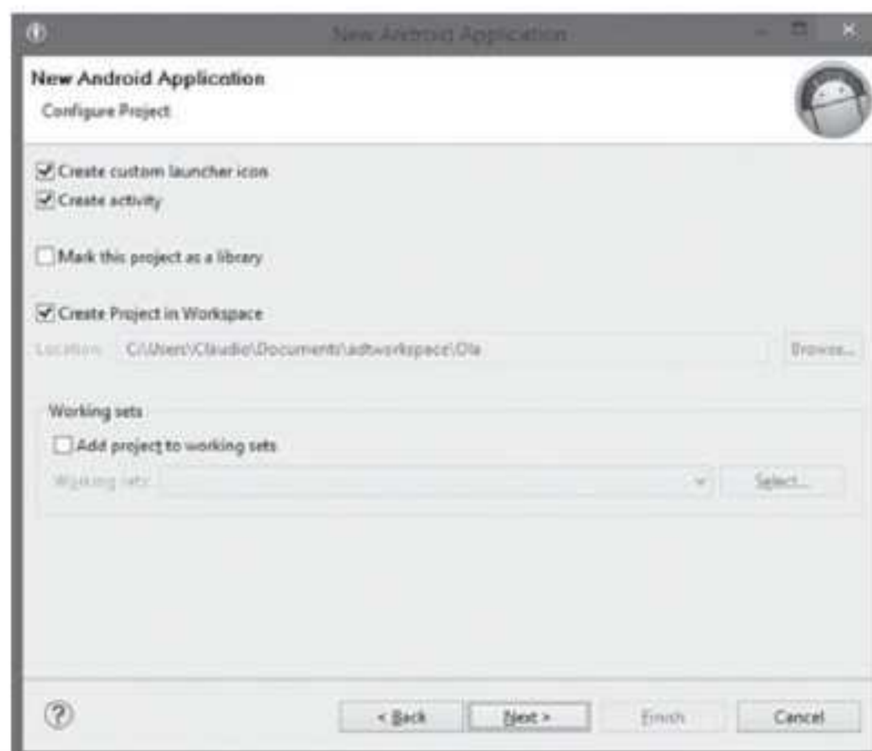


Figura 12.7 – Janela de configuração do projeto.

Em seguida, conforme podemos notar na Figura 12.8, escolha os ícones da aplicação ou aceite os valores sugeridos e pressione novamente o botão "Next >" para continuar com a criação do projeto.



Figura 12.8 – Definição dos ícones do aplicativo.

Nesta próxima tela, marque a opção "Create Activity", selecione a opção "Blank Activity" e pressione o botão "Next >", conforme ilustrado na Figura 12.9.



Figura 12.9 – Criação de uma "Blank Activity".

Defina um nome para a Activity que será criada (ou aceite o valor padrão) e pressione o botão "Finish", conforme mostrado na Figura 12.10.



Figura 12.10 – Definição do nome da Activity.

O projeto do aplicativo será criado, e o código-fonte básico da aplicação e os arquivos de recursos necessários serão automaticamente gerados pelo Eclipse. Agora em "Package Explorer" (Figura 12.11), que está localizado na parte esquerda do Eclipse, identifique as seguintes pastas que fazem parte da estrutura do projeto:

- **src:** contém os arquivos com o código-fonte do aplicativo;
- **res:** possui os arquivos dos recursos utilizados no projeto;

- **res/layout:** contém os arquivos no formato XML com o "desenho" das telas do aplicativo;
- **res/values:** apresenta os arquivos, também em XML, que contêm valores, por exemplo, dos textos a serem utilizados na tela do aplicativo.

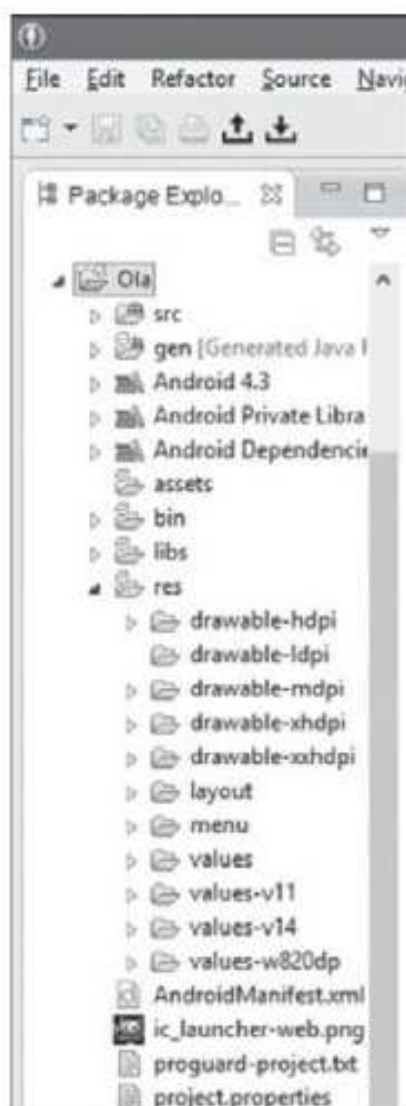


Figura 12.11 – Package Explorer.

Na pasta values localizada dentro da pasta 'res', dê um duplo clique sobre o arquivo strings.xml para abri-lo para edição. Conforme ilustrado na Figura 12.12, altere o valor da string "hello_world" para "Olá Pessoal!".



Figura 12.12 – Editor de Strings.

Em seguida, salve o projeto e execute a aplicação criada pelo ícone "Run" presente na barra de tarefas do Eclipse, conforme mostrado na Figura 12.13.

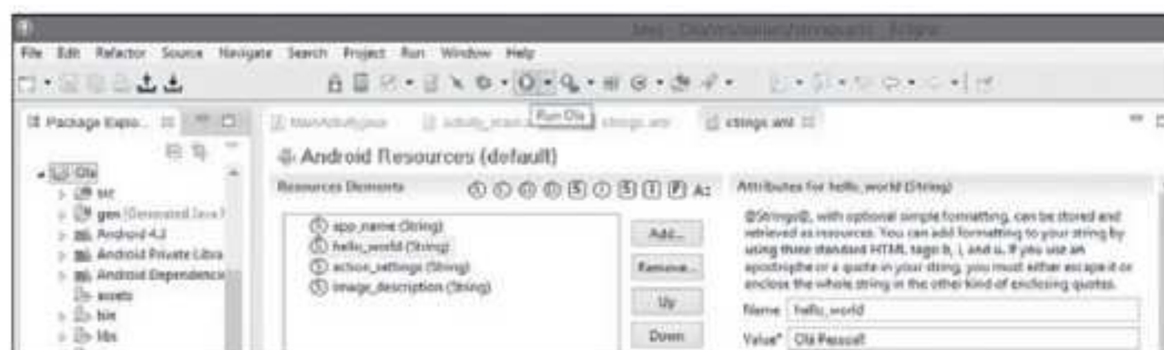


Figura 12.13 – Janela de configuração do projeto.

Observe na Figura 12.14 que a máquina virtual será aberta com a aplicação que foi desenvolvida em execução.

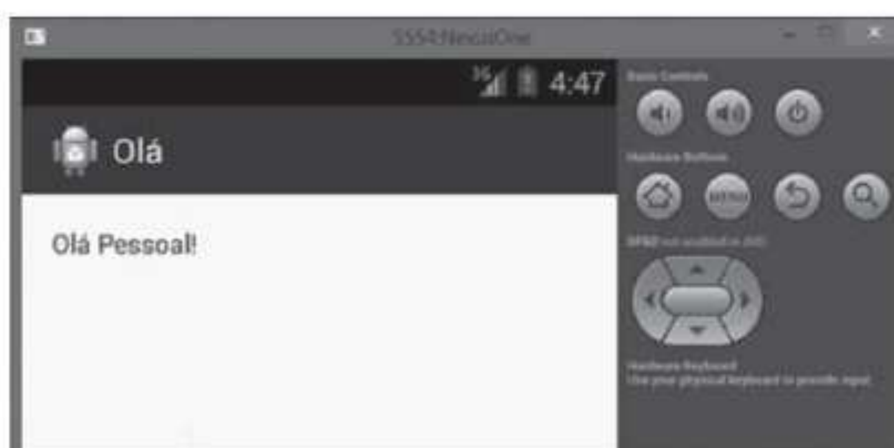


Figura 12.14 – Execução do aplicativo desenvolvido.

12.2 Estrutura básica de um aplicativo Android

Após montarmos um aplicativo bastante simples, vamos utilizá-lo como base para explicarmos a estrutura que um programa Android deve ter. Inicialmente, devemos notar que um aplicativo Android é criado a partir da classe Activity disponível no SDK da plataforma. Assim, a classe principal de um aplicativo deve ser filha da classe Activity, conforme podemos ver a seguir que foi automaticamente gerado quando criamos o projeto no Eclipse.

```
public class MainActivity extendsActivity {
@Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  // Demais métodos gerados
}
```

O método onCreate da classe principal do projeto, neste exemplo MainActivity, usará o setContentView para obter os arquivos XML que descrevem os recursos de tela (/res/layout/activity_main.xml) e textos (/res/values/strings.xml), que também foram gerados automaticamente pelo Eclipse no momento em que o projeto foi criado.

O arquivo activity_main.xml, mostrado a seguir, encontra-se localizado na pasta /res/layout, e contém a descrição dos elementos gráficos e respectivas propriedades que serão mostradas na tela do programa. Note o elemento que define o layout utilizado, neste caso, RelativeLayout, e também o uso do componente TextView, que irá mostrar o texto na tela do aplicativo.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.ola.MainActivity" >

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />

</RelativeLayout>
```

Dentro do Eclipse, em vez de editarmos diretamente o arquivo XML, podemos usar o construtor de interfaces, ilustrado na Figura 12.15, que facilita bastante o processo de desenvolvimento das telas que irão compor o aplicativo.

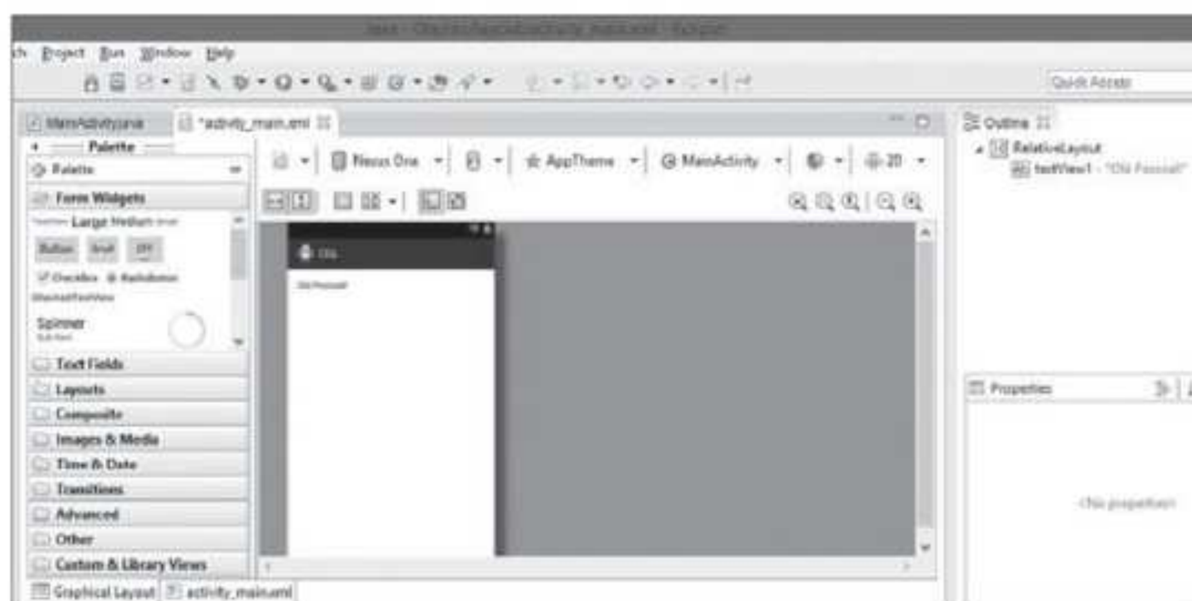


Figura 12.15 – Editor de interfaces do Eclipse.

O outro arquivo XML de recurso que é utilizado é o strings.xml, contido na pasta /res/values. Este arquivo, como podemos observar no código-fonte a seguir, contém a descrição das cadeias de caracteres (strings) utilizadas no programa.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="app_name">Olá</string>
<string name="hello_world">Olá Pessoal!</string>
<string name="action_settings">Configurações</string>

</resources>
```

Com o intuito de facilitar o desenvolvimento dos aplicativos, este arquivo também pode ser configurado graficamente pelo Eclipse, conforme podemos ver na Figura 12.16.

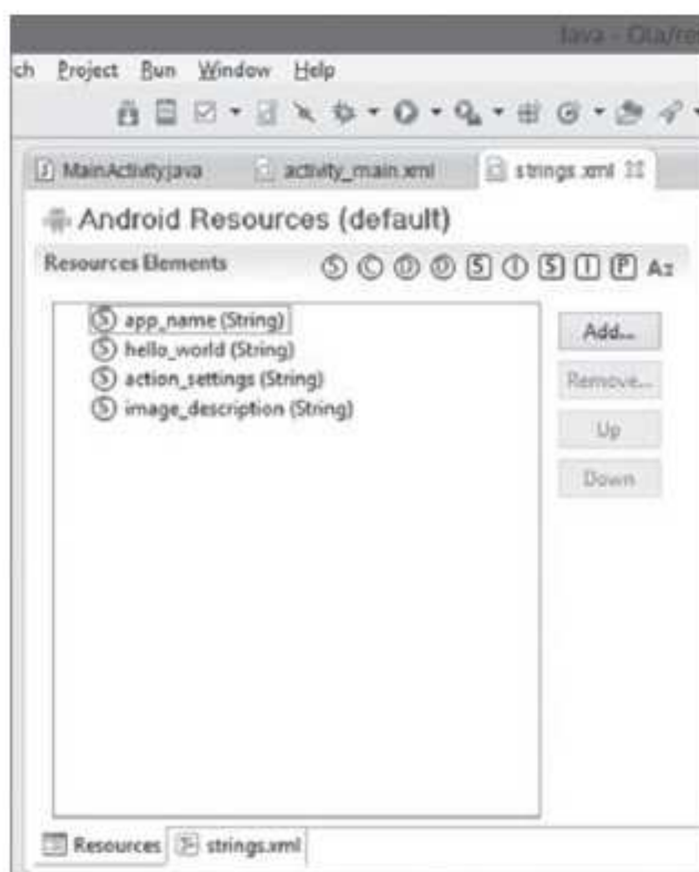


Figura 12.16 – Editor de recursos.

Esses são os elementos básicos que precisamos conhecer para podermos desenvolver os aplicativos que, por meio da comunicação por Bluetooth, irão interagir com os nossos projetos em Arduino.

12.3 Controlando um LED por um aplicativo Android

O conceito básico deste projeto será utilizar a comunicação Bluetooth para que um aplicativo Android controle o acendimento de um LED.



PROJETO Nº 41
Android Bluetooth LED

Neste projeto vamos realizar o módulo Bluetooth HC-05 (ou similar) de modo a permitir que um LED seja aceso ou apagado remotamente por um aplicativo desenvolvido para Android.



Material necessário

- 1 Arduino;
- 1 módulo Bluetooth HC-05 ou similar;
- 1 LED;
- 1 resistor de 220 ohms (vermelho, vermelho, marrom) ou de 330 ohms (laranja, laranja, marrom);
- 1 protoboard;
- jumper cable.



Montagem do circuito

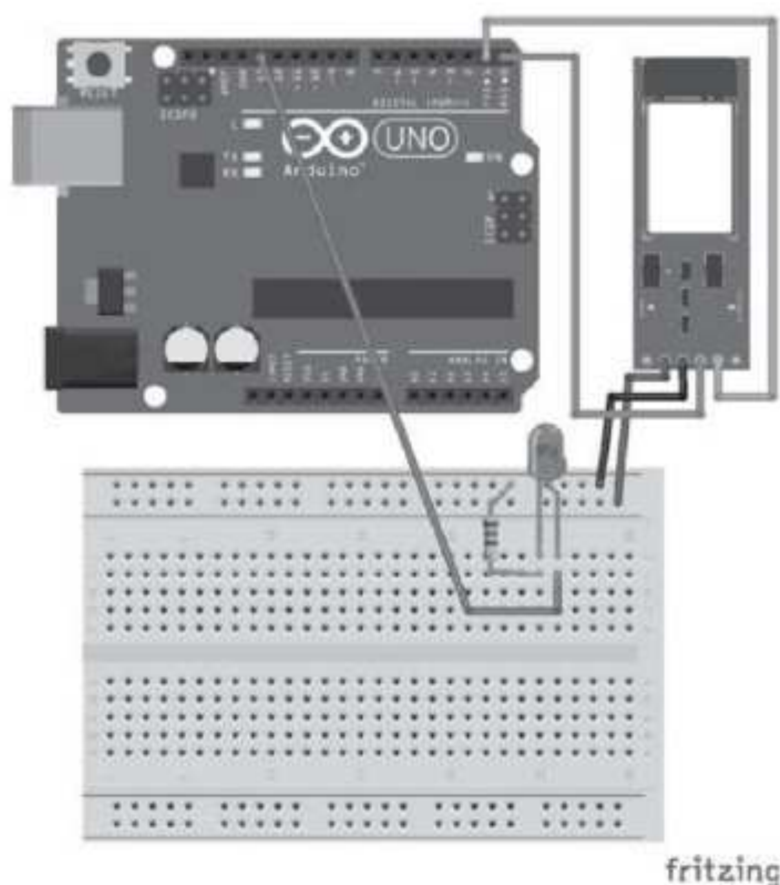


Figura 12.17 – Projeto Bluetooth LED.

Considerando como referência a Figura 12.17, realize a sequência de montagem do projeto:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.
b) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
c) Coloque o resistor de 220 ohms (ou 330 ohms) entre a linha de alimentação negativa e qualquer outra da protoboard.
d) Coloque o LED com o cátodo (lado chanfrado) conectado ao terminal do resistor.
e) Conecte o ânodo do LED ao pino 13 do Arduino.
f) Conecte o pino GND do módulo Bluetooth à linha de alimentação negativa da protoboard.
g) Conecte o pino VCC do módulo Bluetooth à linha de alimentação positiva da protoboard.
h) Conecte o pino de transmissão (TX) do módulo Bluetooth ao pino digital 0 (RX) do Arduino.
i) Conecte o pino de recepção (RX) do módulo Bluetooth ao pino digital 1 (TX) do Arduino.



Programa

Como podemos observar no sketch a seguir, a implementação é bastante simples, visto que o módulo Bluetooth utilizará a comunicação serial:

```
int LED = 13;

void setup() {
  Serial.begin (9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  if (Serial.available()) {
    char caractere = (char) Serial.read();
    if(caractere == '1') {
      digitalWrite(LED, HIGH);
      Serial.println("LED ligado.");
    }
```

```
    else if (caractere == '0') {
      digitalWrite(LED, LOW);
      Serial.println("LED desligado.");
    }
  }
  delay(1000);
}
```

Analisando o código-fonte, podemos observar que quando o caractere '1' é recebido pelo módulo, o LED será aceso, e quando o caractere '0' for recebido, o LED será apagado e qualquer outro caractere será ignorado.



Programa

Agora vamos desenvolver o aplicativo Android que deverá enviar os dados para o nosso projeto no Arduino. No Eclipse, crie um novo "Android Application Project" chamado de ControleLED. Abra o arquivo strings.xml e defina o seguinte conteúdo para ele:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="app_name">Controle de LED</string>
<string name="action_settings">Configurações</string>
<string name="txt_dispositivos">Dispositivos Pareados:</string>
<string name="txt_imagem">LED</string>

</resources>
```

Em seguida, utilizando o editor de interfaces, monte um layout similar ao mostrado na Figura 12.18.



Figura 12.18 – Layout da tela do aplicativo.

Assim, para montar a tela do aplicativo, inicialmente copie uma fotografia ou um desenho de um LED para a pasta /res/drawable-mdpi, conforme ilustrado na Figura 12.19.



Figura 12.19 – Cópia de um arquivo de imagem para a pasta /res/drawable-mdpi.

Em seguida, usando a Paleta, que é mostrada na Figura 12.20, arraste o objeto ImageView para a tela da aplicação.



Figura 12.20 – Paleta de componentes.

Utilize a janela de propriedades para associar a imagem à propriedade Src da ImageView. A imagem copiada estará disponível dentro do item Drawable, conforme podemos notar na Figura 12.21.

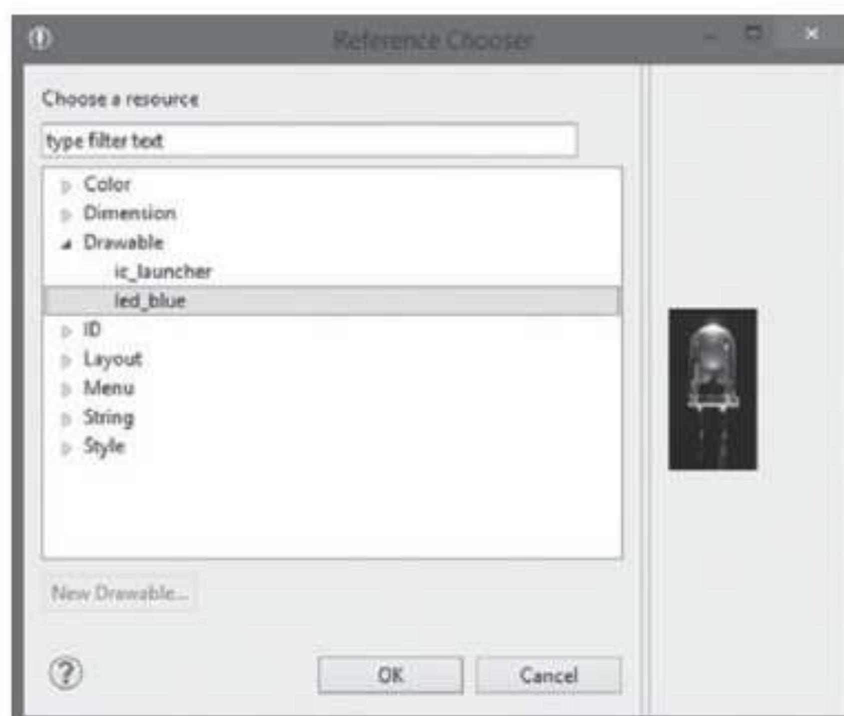


Figura 12.21 – Escolha da referência ao recurso.

Agora procure na Paleta o controle Switch e arraste-o para a tela da aplicação. Defina a propriedade Id do objeto criado para @+id/interruptor. Também pela Paleta crie uma TextView e coloque o valor @string/txt_dispositivos na propriedade Text do objeto. Concluindo o layout da tela do aplicativo, crie, utilizando a Paleta, um objeto a partir do controle Spinner e defina o seu Id como @+id/dispositivo. Após esses passos, a tela do programa estará concluída e, na janela Outline, devemos observar os componentes criados, conforme podemos ver na Figura 12.22.



Figura 12.22 – Estrutura do arquivo de layout (outline).

O funcionamento básico do aplicativo consistirá em o usuário selecionar o dispositivo que está pareado na lista mostrada no objeto Spinner e depois usar o Switch para ligar ou desligar o LED. Lembrar que previamente o dispositivo Android já deverá ter sido pareado com o módulo Bluetooth HC-05 (ou similar).

O passo seguinte consiste em realizar a programação do Switch que foi criado. Para isso, selecione o objeto e digite interruptorClicked na propriedade 'On Click', conforme ilustrado na Figura 12.23.

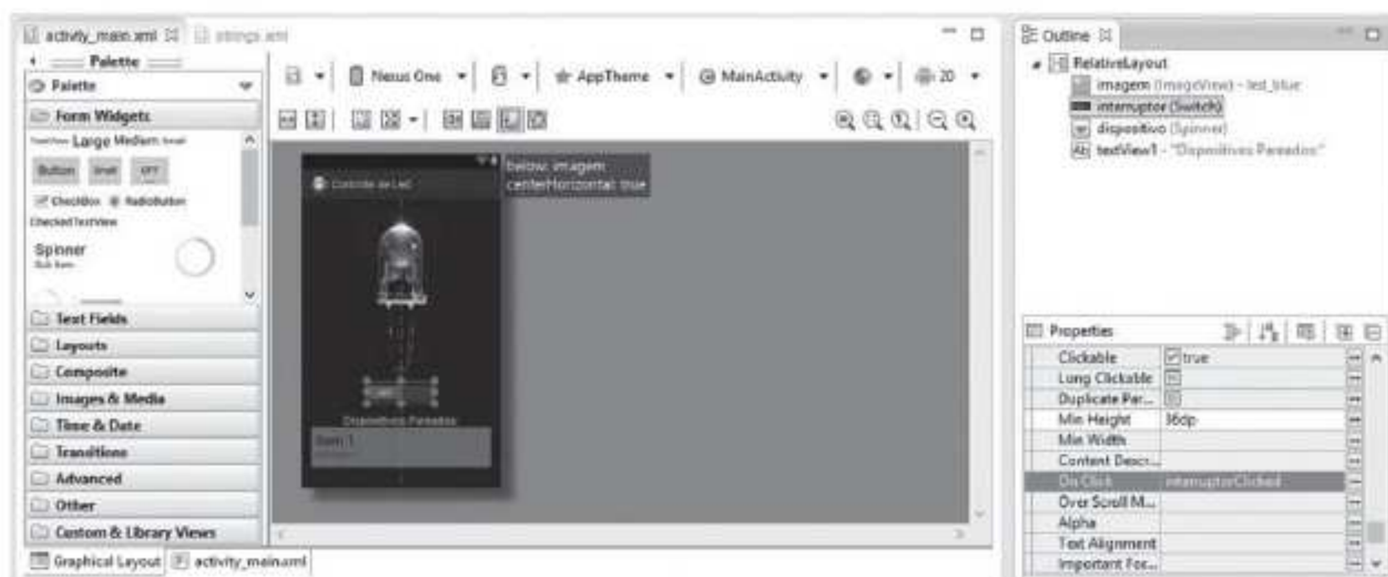


Figura 12.23 – Definição do evento On Click.

Vamos agora passar para a programação do aplicativo. Para isso, identifique na pasta /src o arquivo MainActivity.java e, com um duplo clique, abra-o para edição. Dentro da classe, declare os atributos a seguir:

```
public class MainActivity extends Activity {
private Spinner dispositivo;
  private Switch interruptor;
  private BluetoothAdapter bt =
  BluetoothAdapter.getDefaultAdapter();
  private BluetoothSocket soquete = null;
  private OutputStream saida = null;
  private Set<BluetoothDevice> dispositivosPareados;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    // Restante do código...
  }
}
```

Os atributos criados consistem nos objetos necessários para acessar os elementos da tela, ou seja, o interruptor e o dispositivo. Os demais atributos serão necessários para a comunicação Bluetooth e serão explicados posteriormente. Agora vamos realizar as seguintes alterações no método onCreate:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  interruptor = (Switch) findViewById(R.id.interruptor);
  dispositivo = (Spinner) findViewById(R.id.dispositivo);
```

```
  ArrayAdapter<String> dados = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item);
  dados.add("Selecione um dispositivo");

  // Verificar se o aparelho tem suporte a Bluetooth
  if (bt != null) {
    // Se o Bluetooth não estiver ligado, criar uma
    // intent para ativá-lo
    if (!bt.isEnabled()) {
      Intent enableBtIntent =
new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      int REQUEST_ENABLE_BT = 1;
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }

    dispositivosPareados = bt.getBondedDevices();
    if (dispositivosPareados.size() > 0) {
      for (BluetoothDevice item : dispositivosPareados) {
        dados.add(item.getName());
      }
    }
  }

  dados.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);
  dispositivo.setAdapter(dados);
}
```

Como podemos notar, no método onCreate, criamos uma Intent (intenção) que habilitará o Bluetooth do dispositivo caso este se encontre desligado. Em seguida, obtemos uma lista contendo os dispositivos pareados por meio do método getBondedDevices e carregamos essa lista no objeto Spinner (dispositivo), utilizando um ArrayAdapter.

Na classe MainActivity, vamos inserir um novo método, que chamamos criarSoqueteBluetooth e que se encontra detalhado a seguir. Ele estabelecerá um Socket (conexão) entre os dois dispositivos Bluetooth, ou seja, o Android e o módulo HC-05 (ou similar).

```
private BluetoothSocket criarSoqueteBluetooth(
BluetoothDevice dispositivo) throws IOException {
  Method método;
  BluetoothSocket tmpSoquete = null;
```

```
  try {
    método =
      dispositivo.getClass().getMethod("createRfcommSocket",
      new Class[] {int.class});
    tmpSoquete = (BluetoothSocket) metodo.invoke(dispositivo, 1);
  }
  catch (SecurityException e) {}
  catch (NoSuchMethodException e) {}
  catch (IllegalArgumentException e) {}
  catch (IllegalAccessException e) {}
  catch (InvocationTargetException e) {}
  return tmpSoquete;
}
```

Concluindo o aplicativo, vamos desenvolver o método interruptorClicked que será utilizado para acender ou apagar o LED.

```
public void interruptorClicked(View v) {
  String dados = "0";
  if (interruptor.isChecked()) {
    dados = "1";
  }
  Toast mensagem = Toast.makeText(this.getApplicationContext(),
    "Enviando para " +
    dispositivo.getSelectedItem().toString() +
    (dados.equals("1")? " Ligar": " Desligar"),
  Toast.LENGTH_LONG);
  mensagem.show();

  if (dispositivosPareados.size() > 0) {
    for (BluetoothDevice item : dispositivosPareados) {
      if (dispositivo.getSelectedItem().toString()
        .equalsIgnoreCase(item.getName())) {
        try {
          BluetoothDevice dispostivoRemoto =
            bt.getRemoteDevice(item.getAddress());
          soquete = criarSoqueteBluetooth(dispostivoRemoto);
          soquete.connect();

          // Cancelar a descoberta, pois torna a conexão lenta
          bt.cancelDiscovery();
          saida = soquete.getOutputStream();
```

```
          byte[] buffer = dados.getBytes();
          saida.write(buffer);
          saida.close();
          soquete.close();
        }
        catch (IOException erro) {}
      }
    }
  }
}
```

Analisando o código-fonte implementado no método, notamos, em primeiro lugar, que definimos se iremos enviar o caractere '0' ou '1' conforme a posição do interruptor (Switch):

```
String dados = "0";
if (interruptor.isChecked()) {
dados = "1";
}
```

Em seguida, utilizando um objeto instanciado da classe Toast, colocamos uma mensagem na tela do dispositivo informando o envio dos dados, conforme mostra o trecho de programa:

```
Toast mensagem = Toast.makeText(this.getApplicationContext(),
  "Enviando para " +
  dispositivo.getSelectedItem().toString() +
  (dados.equals("1")? " Ligar": " Desligar"),
  Toast.LENGTH_LONG);
mensagem.show();
```

Concluindo o método, os dados são enviados pela conexão (soquete) estabelecida entre os dois dispositivos:

```
BluetoothDevice dispositivoRemoto =
  bt.getRemoteDevice(item.getAddress());
soquete = criarSoqueteBluetooth(dispostivoRemoto);
soquete.connect();

// Cancelar a descoberta, pois torna a conexão lenta
bt.cancelDiscovery();
saída = soquete.getOutputStream();
```

```
byte[] buffer = dados.getBytes();
saída.write(buffer);
saída.close();
soquete.close();
```

O aplicativo está pronto para ser utilizado, porém, antes disso, acrescente as linhas a seguir no arquivo AndroidManifest.xml, antes do item <application ...> já existente:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Essas instruções são as permissões necessárias para que o aplicativo possa acessar o Bluetooth do dispositivo Android. Para testar o programa, transfira o programa Android para um smartphone ou tablet, ligue o projeto desenvolvido com o Arduino e pareie os dois dispositivos. Depois, basta executar a aplicação Android e notar que quando você toca sobre o botão liga-desliga (Switch) que está na tela, o LED muda o seu estado de apagado para aceso ou vice-versa.

12.4 Matriz de LEDs controlada por Bluetooth

Aplicando os mesmos conceitos abordados no aplicativo ControleLED, vamos desenvolver um novo projeto que demonstra as possibilidades de integração entre o Arduino e o Android.



PROJETO Nº 42
Matriz de LEDs controlada por Bluetooth

Neste projeto vamos adaptar o projeto da matriz de LEDs, desenvolvido anteriormente, para realizar a comunicação Bluetooth, de modo a permitir que ela seja controlada por um outro dispositivo que poderá enviar mensagens, controlar a velocidade de exibição e pausar o texto.



Material necessário

- 1 Arduino;
- 1 módulo Bluetooth HC-05 ou similar;
- 2 matrizes de LED de 8x8*;
- 2 CI MAX 7219 ou 7221*;

- 2 resistores de 100 kohms (marrom, preto, amarelo)*;
- 1 protoboard;
- jumper cable.

* Podem ser substituídos por dois módulos de matriz de LEDs.



Montagem do circuito

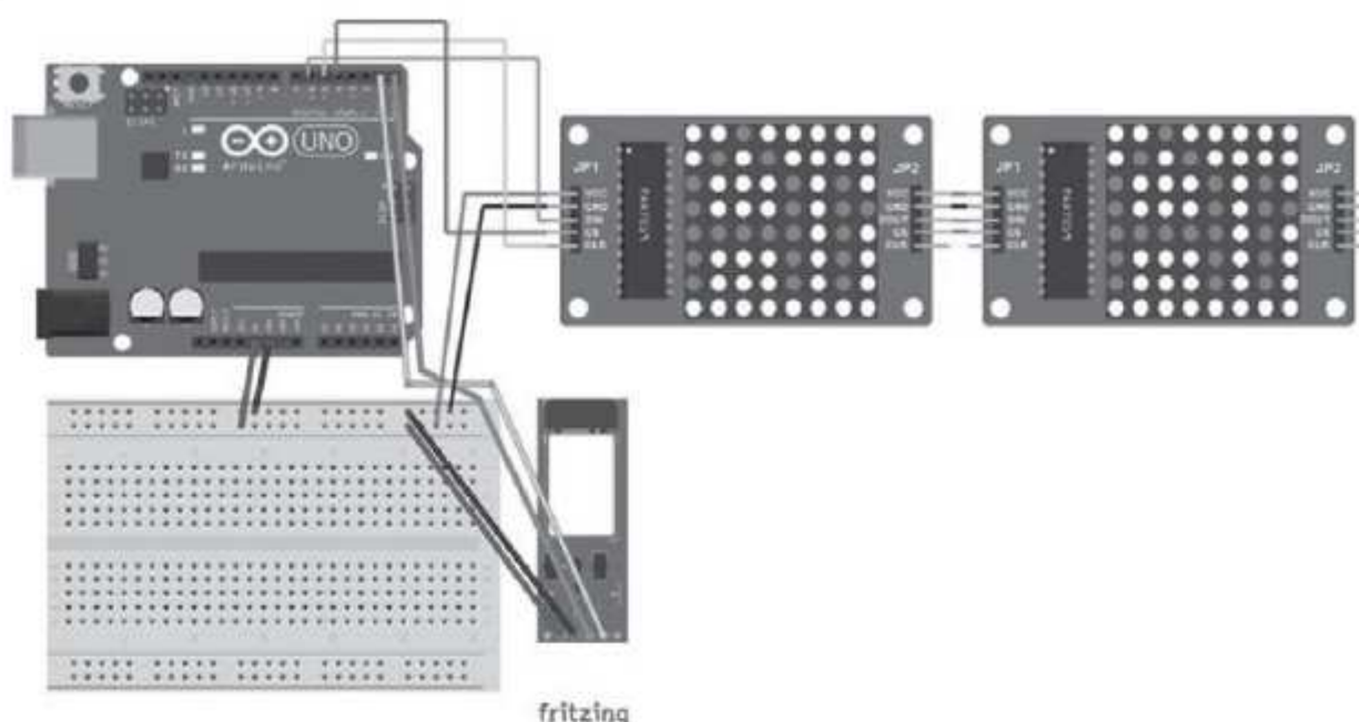


Figura 12.24 – Uso do Bluetooth para controlar matrizes de LEDs.

Considerando como referência a Figura 12.24, realize a sequência de montagem:

a) Conecte o pino GND do Arduino à linha de alimentação negativa (preta ou azul) da protoboard.
b) Conecte o pino 5 V do Arduino à linha de alimentação positiva (vermelha) da protoboard.
c) Conecte o pino GND do módulo Bluetooth à linha de alimentação negativa da protoboard.
d) Conecte o pino VCC do módulo Bluetooth à linha de alimentação positiva da protoboard.
e) Coloque o pino de transmissão (TX) do módulo Bluetooth ao pino digital 0 (RX) do Arduino.
f) Conecte o pino de recepção (RX) do módulo Bluetooth ao pino digital 1 (TX) do Arduino.
g) Conecte o pino VCC do primeiro módulo de matriz à linha de alimentação positiva (5 V) da protoboard.

h) Conecte o pino GND do primeiro módulo de matriz à linha de alimentação negativa da protoboard.

i) Conecte o pino DIN do primeiro módulo de matriz ao pino 6 do Arduino.

j) Conecte o pino CS (LOAD/CS) do primeiro módulo de matriz ao pino 4 do Arduino.

k) Conecte o pino CLK do primeiro módulo de matriz ao pino 5 do Arduino.

l) Conecte o pino VCC do primeiro módulo de matriz ao pino VCC do segundo módulo de matriz.

m) Conecte o pino GND do primeiro módulo de matriz ao pino GND do segundo módulo de matriz.

n) Conecte o pino DOUT do primeiro módulo de matriz ao pino DIN do segundo módulo de matriz.

o) Conecte o pino CS (LOAD/CS) do primeiro módulo de matriz ao pino CS do segundo módulo de matriz.

p) Conecte o pino CLK do primeiro módulo de matriz ao pino CLK do segundo módulo de matriz.



Programa

Digite o sketch a seguir no ambiente de desenvolvimento do Arduino, lembrando-se de inserir no projeto as bibliotecas necessárias, ou seja, avr/pgmspace.h, LEDControl.h e FonteMatriz.h.

```
#include <avr/pgmspace.h>
#include "LEDControl.h"
#include "FonteMatriz.h"

const int NUM_MATRIZ = 2;

/*
 * Criar um LEDControl (matrizLED).
 * O pino 6 do Arduino deve ser conectado ao pino DATA IN do
 * primeiro MAX 7219/21
 * O pino 5 do Arduino deve ser conectado ao pino CLK do primeiro
 * MAX 7219/21
 * O pino 4 do Arduino deve ser conectado ao pino LOAD (/CS) do
 * primeiro MAX 7219/21
 * O quarto parâmetro indica que há apenas um MAX 7219/21
 * conectado ao Arduino (NUM_MATRIZ)
 */
LEDControl lc=LEDControl(6, 5, 4, NUM_MATRIZ);
```

```
String linha = "";
char comando = 'R';
String mensagem = "Envie um texto.";
int atraso = 500;

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < NUM_MATRIZ; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 2);
    lc.clearDisplay(i);
  }
  }

boolean obterComando() {
  linha = "";
  while (Serial.available() > 0) {
    // Ler o byte que está na entrada serial
    linha += (char) Serial.read();
  }
  if (!linha.equals("")) {
    comando = linha.charAt(0);

    if (comando == 'R') {
      if (linha.substring(1, linha.length()).toInt() > 0)
        atraso = linha.substring(1, linha.length()).toInt();
    }
    else if (comando == 'T') {
      if (!linha.substring(1, linha.length()).equals(""))
        mensagem = linha.substring(1, linha.length());
    }
    return (true);
  }
  else
    return (false);
}

void loop() {
  int contador = 0;
  int caractere = 0;
  int caractereAnterior = 0;
```

```
  do {
    // Verificar a porta serial
    if (obterComando() && comando == 'T')
      break;

    if (comando != 'P') {
      // Obtém cada caractere que compõe a mensagem
      caractere =  mensagem.charAt(contador);
      if (caractere != 0) {
        exibirCaractere(1, caractere);
        exibirCaractere(0, caractereAnterior);
        caractereAnterior = caractere;
        delay(atraso);
      }
      contador++;
    }
  }
  while (caractere != 0);
  lc.clearDisplay(0);
}

void exibirCaractere(int matriz, int ascii) {
  if (ascii >= 0x20 && ascii <= 0x7f) {
    // Exibe as sete linhas que compõem o caractere
    for (int i = 0; i < 7; i++) {
      // Busca na tabela de caracteres uma linha
      int linha =
        pgm_read_byte_near(fonte5x7 + ((ascii - 0x20) * 8) + i);
      lc.setColumn(matriz, 7 - i, linha);
    }
  }
}
```

Você pode observar que o sketch que irá executar no Arduino é o mesmo que foi utilizado no Projeto nº 38 (envio de textos para uma matriz de LEDs), desenvolvido no Capítulo 11.

Desta forma, o programa receberá, a partir da porta serial à qual está conectado o módulo Bluetooth, uma string cujo primeiro caractere define a função a ser executada, ou seja, "R" para rolar o texto em determinada velocidade, "T" para definir o texto a ser mostrado nas matrizes e "P" para pausar a rolagem.



Programa

Vamos agora passar para o desenvolvimento do aplicativo Android que controlará as matrizes de LEDs, ou seja, enviará uma cadeia de caracteres pela conexão Bluetooth ao Arduino. No Eclipse, crie um novo "Android Application Project" chamado de MatrizLED.

Inicialmente, edite o arquivo strings.xml para definir os textos que serão utilizados pelo aplicativo conforme ilustrado a seguir, lembrando também que você pode utilizar o construtor de interface do Eclipse para realizar a mesma tarefa.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="app_name">Matriz de LEDs</string>
<string name="msg_enviar">Por favor, digite o texto que deseja exibir na matriz
de LEDs:</string>
<string name="action_settings">Configurações</string>
<string name="msg_dispositivo">Dispositivos pareados:</string>
<string name="msg_velocidade">Velocidade:</string>
<string name="msg_texto">Enviar o texto</string>
<string name="msg_rolar">Rolar</string>
<string name="msg_pausar">Pausar</string>

</resources>
```

Procure três imagens no formato png, com dimensões próximas a 64 x 64 pixels, que serão utilizadas nos botões definir texto, rolar o texto e pausar a rolagem (Figura 12.25).



Figura 12.25 – Sugestão de imagens para os botões do aplicativo.

Após a definição dos recursos de texto e imagem, agora passamos ao arquivoactivity_main.xml no qual iremos construir o layout da tela do nosso aplicativo. Dessa forma, defina o seguinte conteúdo para o arquivo:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
```

```
android:background="@android:color/background_light"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.example.matrizLED.MainActivity" >

<TextView
  android:id="@+id/textView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:text="@string/msg_enviar"
  android:textSize="18sp" />

<EditText
  android:id="@+id/texto"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/textView1"
  android:layout_alignRight="@+id/textView1"
  android:layout_below="@+id/textView1"
  android:layout_centerHorizontal="true"
  android:ems="10"
  android:inputType="textMultiLine"
  android:minLines="3" />

<TextView
  android:id="@+id/textView3"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/texto"
  android:layout_below="@+id/texto"
  android:layout_marginTop="28dp"
  android:text="@string/msg_velocidade"
  android:textSize="18sp" />

<SeekBar
  android:id="@+id/velocidade"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/textView3"
  android:layout_below="@+id/textView3"
  android:max="2000"
  android:progress="1500" />
```

```
<TextView
  android:id="@+id/textView2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/velocidade"
  android:layout_alignLeft="@+id/velocidade"
  android:layout_marginTop="28dp"
  android:text="@string/msg_dispositivo"
  android:textSize="18sp" />

<Spinner
  android:id="@+id/dispositivo"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/textView2"
  android:layout_alignLeft="@+id/textView2" />

<ImageButton
  android:id="@+id/enviar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/dispositivo"
  android:layout_alignTop="@+id/rolar"
  android:background="@android:color/background_light"
  android:contentDescription="@string/msg_enviar"
  android:onClick="enviarClicked"
  android:src="@drawable/botao_texto" />

<ImageButton
  android:id="@+id/rolar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@+id/dispositivo"
  android:layout_marginLeft="16dp"
  android:layout_marginTop="28dp"
  android:layout_toRightOf="@+id/enviar"
  android:background="@android:color/background_light"
  android:contentDescription="@string/msg_rolar"
  android:onClick="rolarClicked"
  android:src="@drawable/botao_rolar" />

<ImageButton
  android:id="@+id/pausar"
  android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/rolar"
    android:layout_marginLeft="16dp"
    android:layout_toRightOf="@+id/rolar"
    android:background="@android:color/background_light"
    android:contentDescription="@string/msg_pausar"
    android:onClick="pausarClicked"
    android:src="@drawable/botao_pausar" />

</RelativeLayout>
```

Após a definição do layout, a tela deverá ser similar à apresentada na Figura 12.26.



Figura 12.26 – Tela do aplicativo.

Note que o texto que será enviado para as matrizes de LEDs será digitado, pelo usuário, em um EditText. O controle da velocidade será efetuado por meio de uma SeekBar, e a seleção do dispositivo Bluetooth pareado ocorrerá em um Spinner. Os últimos botões terão a finalidade de enviar uma cadeia de caracteres com os comandos e dados por meio da conexão Bluetooth.

Após a construção da tela do nosso aplicativo, iniciaremos o desenvolvimento das rotinas que realizarão a conexão a outro dispositivo Bluetooth pareado e o envio dos dados. O primeiro passo será declarar os atributos que serão necessários. Assim, conforme podemos observar no trecho de programa a seguir, dentro da classe MainActivity defina os atributos para os elementos EditText, SeekBar e Spinner e os que serão necessários para a utilização do Bluetooth.

```
public class MainActivity extends Activity {
  // Insira os atributos a seguir
  private EditText texto;
  private SeekBar velocidade;
  private Spinner dispositivo;
  private BluetoothAdapter bt =
    BluetoothAdapter.getDefaultAdapter();
  private BluetoothSocket soquete = null;
  private OutputStream saida = null;
  private Set<BluetoothDevice> dispositivosPareados;

  // Não apague ou altere o restante do código-fonte
```

No método onCreate, mostrado a seguir, vamos associar os atributos texto, dispositivo e velocidade aos respectivos elementos usados na tela do aplicativo. Para isso, utilizamos o método findViewById. Na sequência, verificamos se existe suporte ao Bluetoth no dispositivo Android. Caso afirmativo, verificamos, pelo método isEnabled, se é necessário ligá-lo.

Para ligar o Bluetooth devemos definir uma Intent, ou seja, uma intenção. No Android, sempre que precisamos acessar um recurso do aparelho, devemos fazê-lo por uma Intent, pois o recurso pode estar não disponível no momento; por exemplo, pode estar sendo utilizado por outro aplicativo.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  texto = (EditText) findViewById(R.id.texto);
  dispositivo = (Spinner) findViewById(R.id.dispositivo);
  velocidade = (SeekBar) findViewById(R.id.velocidade);

  ArrayAdapter<String> dados = new ArrayAdapter<String>(this,
  android.R.layout.simple_spinner_item);
  dados.add("Selecione um dispositivo");

  // Verificar se o aparelho tem suporte a Bluetooth
  if (bt != null) {
    // Se o Bluetooth não estiver ligado, criar uma Intent
    // para ativá-lo
    if (!bt.isEnabled()) {
      Intent enableBtIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
```

```
      int REQUEST_ENABLE_BT = 1;
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }

    dispositivosPareados = bt.getBondedDevices();
    if (dispositivosPareados.size() > 0) {
      for (BluetoothDevice item: dispositivosPareados) {
        dados.add(item.getName());
      }
    }
  }

  dados.setDropDownViewResource(
  android.R.layout.simple_spinner_dropdown_item);
  dispositivo.setAdapter(dados);
}
```

Após essas verificações, devemos obter uma lista dos dispositivos que estão pareados. A lista é obtida pelo método getBondedDevices, e, em seguida, devemos carregá-la no objeto dispositivos (Spinner), ou seja:

```
dispositivosPareados = bt.getBondedDevices();
if (dispositivosPareados.size() > 0) {
  for (BluetoothDevice item: dispositivosPareados) {
    dados.add(item.getName());
  }
}
```

O método criarSoqueteBluetooth, mostrado a seguir, deverá ser inserido dentro da classe da MainActivity e será responsável por criar um soquete (conexão) entre o dispositivo Android e aquele que foi selecionado no Spinner.

```
private BluetoothSocket criarSoqueteBluetooth(
  BluetoothDevice dispositivo) throws IOException {
  Method método;
  BluetoothSocket tmpSoquete = null;
  try {
    método = dispositivo.getClass()
      .getMethod("createRfcommSocket", new Class[] {int.class});
    tmpSoquete = (BluetoothSocket) metodo.invoke(dispositivo, 1);
  }
  catch (SecurityException e) {}
  catch (NoSuchMethodException e) {}
```

```
  catch (IllegalArgumentException e) {}
  catch (IllegalAccessException e) {}
  catch (InvocationTargetException e) {}
  return tmpSoquete;
}
```

O método enviarDadosBluetooth irá enviar uma cadeia de caracteres por meio da conexão que foi criada pelo método criarSoqueteBluetooth. Para fazer isso, devemos "escrever" a cadeia de caracteres a ser enviada no atributo saída, que é um objeto da classe OutputStream, ou seja, permite enviar um fluxo de dados por um canal de distribuição, o qual, neste exemplo, é o soquete (conexão) Bluetooth.

```
private void enviarDadosBluetooth(String dados) {
  Toast mensagem = Toast.makeText(this.getApplicationContext(),
    "Enviando para " +
    dispositivo.getSelectedItem().toString(), Toast.LENGTH_LONG);
  mensagem.show();

  if (dispositivosPareados.size() > 0) {
    for (BluetoothDevice item: dispositivosPareados) {
      if (dispositivo.getSelectedItem().toString()
        .equalsIgnoreCase(item.getName())) {
        try {
          BluetoothDevice dispositivoRemoto =
          bt.getRemoteDevice(item.getAddress());
          soquete = criarSoqueteBluetooth(dispositivoRemoto);
          soquete.connect();

          // Cancelar a descoberta, pois torna a conexão
          // mais lenta
          bt.cancelDiscovery();

          saída = soquete.getOutputStream();
          byte[] buffer = dados.getBytes();
          saída.write(buffer);
          saída.close();
          soquete.close();
        }
        catch (IOException erro) {
        }
      }
    }
  }
}
```

A seguir realizaremos a programação do evento "On Click" do botão enviar. Devemos iniciar a cadeia de caracteres que será enviada, pelo método enviarDadosBluetooth com a letra "T", e, em seguida, concatenar a mensagem que foi digitada pelo usuário no campo texto.

```
public void enviarClicked(View v) {
  enviarDadosBluetooth ("T" + texto.getText().toString());
}
```

O botão rolar enviará, pela conexão Bluetooth, o comando "R" seguido pela velocidade com que a rolagem deverá ocorrer no Arduino, conforme podemos observar no método definido a seguir:

```
public void rolarClicked(View v) {
  enviarDadosBluetooth ("R" + (velocidade.getProgress());
}
```

Concluindo o desenvolvimento do aplicativo, o clique no botão pausar irá fazer com que o comando "P" seja enviado para o Arduino, ou seja:

```
public void pausarClicked(View v) {
  enviarDadosBluetooth ("P");
}
```

Compile a aplicação e depois execute-a em um aparelho Android com Bluetooth. Realize o pareamento entre os dois dispositivos e observe que, ao clicar nos botões, você estará enviando os dados para o Arduino, no qual ele realizará o controle das matrizes de LEDs.

Exercícios

1. Altere o Projeto nº 41 (Android Bluetooth LED) para acender ou apagar individualmente cada componente de cor de um LED RGB.
2. Altere o projeto Projeto nº 42 (matriz de LEDs controlada por Bluetooth) acrescentando ao programa Android um botão de Reset. Ao pressionar esse botão, o texto a ser exibido na matriz de LEDs deverá ser alterado para "Envie um texto" e o atraso deverá ser ajustado para 500.

13 Documentando seus Projetos com Fritzing

À medida que criamos novos projetos ou adaptamos projetos já existentes para desempenhar novas funções, é necessário armazenar, além do código-fonte dos sketches, os diagramas dos circuitos montados. O Fritzing é uma ferramenta muito útil para desenharmos os diagramas (esquemas) dos projetos desenvolvidos. Ele, por exemplo, foi utilizado para montar os esquemas apresentados neste livro.

Neste capítulo apresentaremos um tutorial básico para que você entenda o funcionamento da ferramenta e também possa guardar os diagramas dos projetos que desenvolverá.

13.1 Download e instalação

O Fritzing pode ser baixado gratuitamente no endereço http://www.fritzing.org, no link "Downloads". A ferramenta não precisa ser instalada, basta descompactar o arquivo em uma pasta e, a partir daí, executar o arquivo "Fritzing.exe". Também é uma boa ideia criar um atalho para o programa na área de trabalho, conforme ilustra a Figura 13.1.

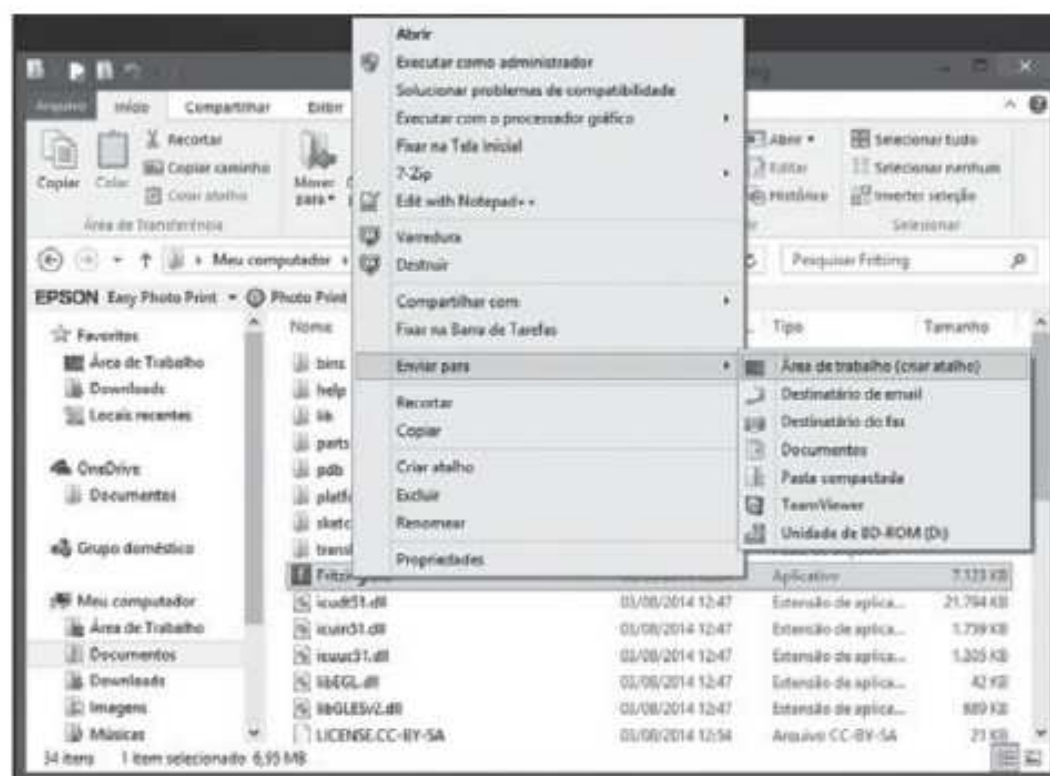


Figura 13.1 – Criar um atalho na área de trabalho.

13.2 Tutorial

Após abrir o Fritzing, devemos primeiramente nomear e salvar o nosso esquema. É importante sempre salvar seu projeto durante o processo de desenho do diagrama. Para fazer isso, acesse a barra de menu e selecione "Arquivo" e depois "Salvar como". Uma nova janela será aberta, e podemos especificar o nome do diagrama e o local (pasta) no qual ele será armazenado. Após fornecer essas informações, clique no botão "Salvar", conforme podemos observar na Figura 13.2.

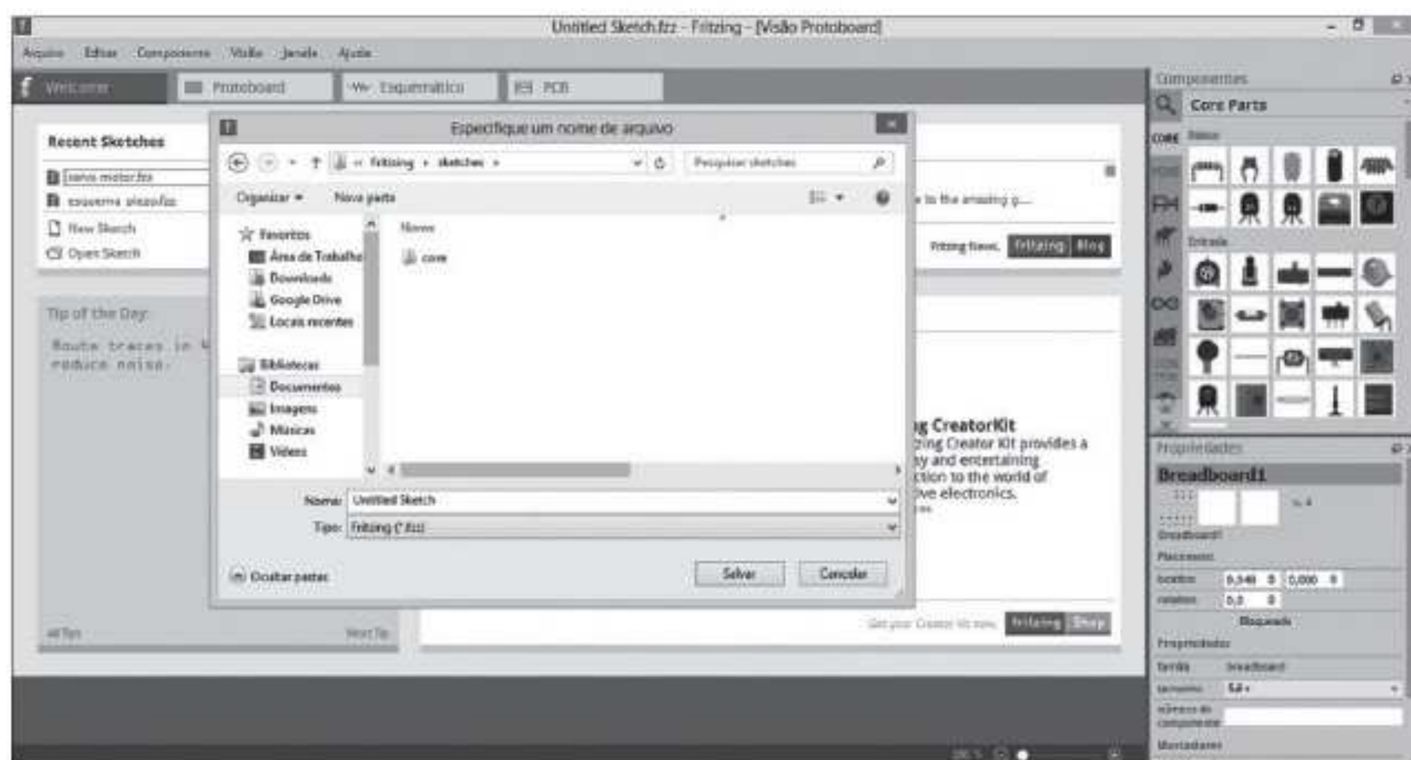


Figura 13.2 – Armazenar um diagrama.

Quando quiser criar um novo diagrama, selecione no menu a opção "Arquivo" e, em seguida, "Novo".

Como podemos notar na Figura 13.3, a versão 4 do Fritzing apresenta as seguintes visões para a área de trabalho:

- **Welcome:** exibe quais são os projetos salvos mais recentemente, dicas e notícias do blog do projeto Fritzing, entre outras informações. Serve apenas como uma tela principal e de apresentação.
- **Protoboard:** é utilizada para montagem da protoboard. Essa visão da área de trabalho foi utilizada para montar os projetos presentes neste livro. Nessa visão, colocamos nossa placa Arduino, os componentes na protoboard e realizamos todas as conexões. Geralmente, é por aqui que começamos nosso projeto, e consiste na visão mais usada no desenho dos diagramas.
- **Esquemático:** mostra o esquema do circuito eletrônico extraído a partir da montagem que foi realizada na visão protoboard. Esta é indicada para aqueles que querem conhecer especificações técnicas da parte eletrônica do projeto.

- **PCB:** permite desenhar e exportar o desenho necessário para a produção de uma placa de circuito impresso à montagem definitiva do projeto. É uma visão restrita aos usuários com conhecimento técnico avançado em eletrônica e que irão confeccionar a placa de circuito impresso.

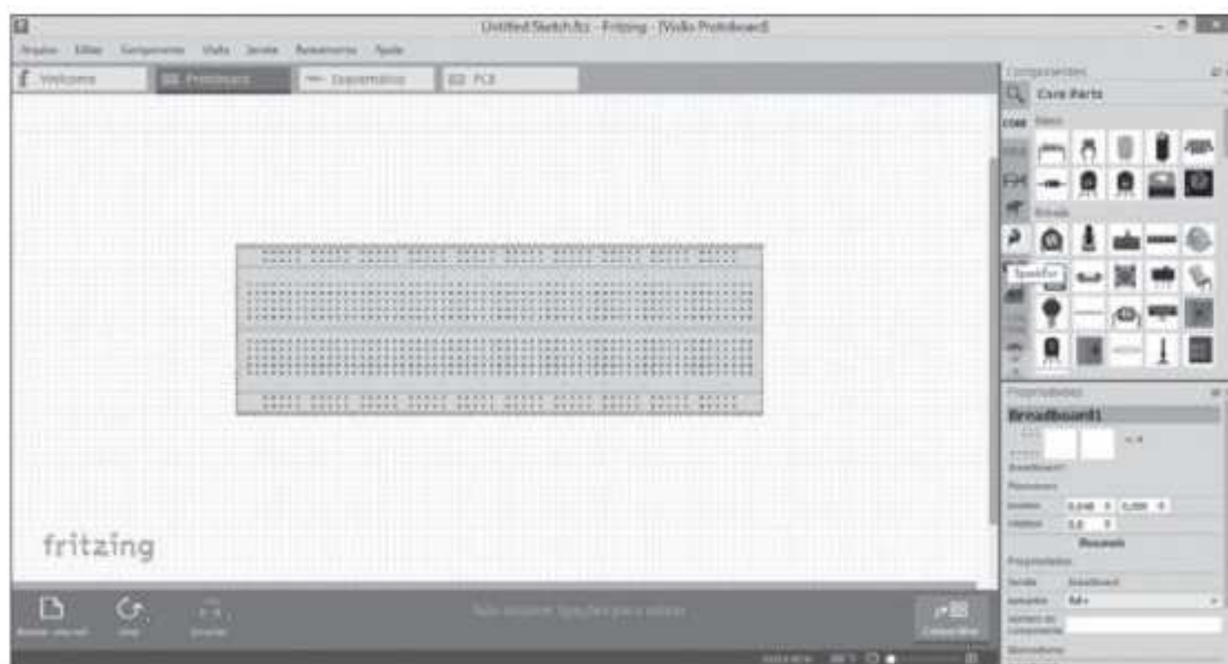


Figura 13.3 – Visões para a área de trabalho.

Conforme ilustrado na Figura 13.3, quando iniciamos um novo diagrama, a visão protoboard já se inicia com a presença de uma protoboard no centro da área de trabalho. No lado esquerdo da janela encontramos a paleta de componentes e logo abaixo, a de Propriedades. Na paleta de componentes, podemos selecionar e arrastar (drag&drop) um componente para a protoboard, assim como a placa Arduino que será utilizada. Essa paleta está organizada em abas, como a "Core", que possui os componentes básicos, e a aba com o símbolo do Arduino, incluindo vários modelos de placas. Para iniciar a montagem do projeto, arraste a placa Arduino que será utilizada até a área de trabalho, logo acima da protoboard (Figura 13.4).

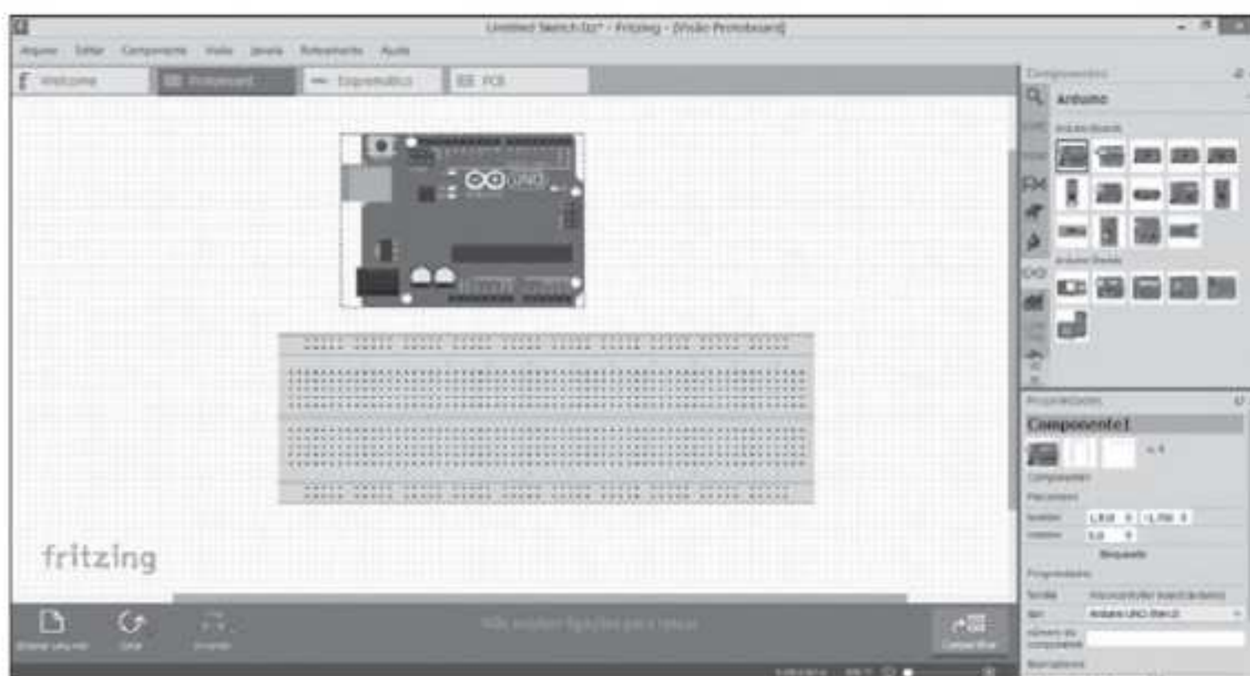


Figura 13.4 – Inserção de uma placa Arduino ao esquema.

Procure na aba "Core" um LED e um resistor que iremos utilizar neste exemplo. Arraste cada um deles para a protoboard. Você notará que o Fritzing auxilia a visualização da conexão do componente, demarcando os furos com um contorno esverdeado, a linha dessa conexão. Observe a Figura 13.5 e realize a inserção dos componentes como se estivesse montando o circuito na protoboard real.

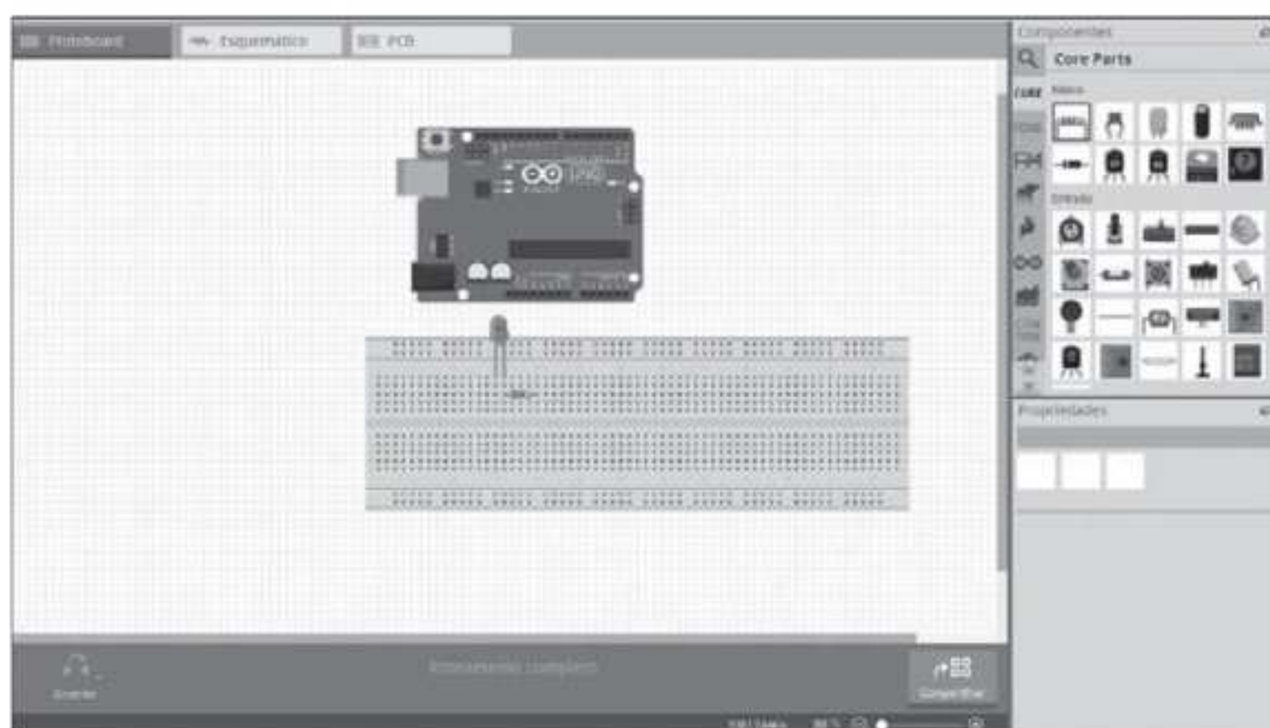


Figura 13.5 – Posicionando um LED e um resistor na protoboard.

Esses componentes podem ter variações, ou seja, modelos e especificações diversas, como é o caso do LED e resistor. Quando um componente possuir características e especificações diversificadas, ao clicar nele será habilitada a paleta de Propriedades, que está localizada abaixo da Componentes. Neste exemplo, ilustrado na Figura 13.6, selecione o LED que foi colocado na protoboard e altere a sua propriedade cor para verde.

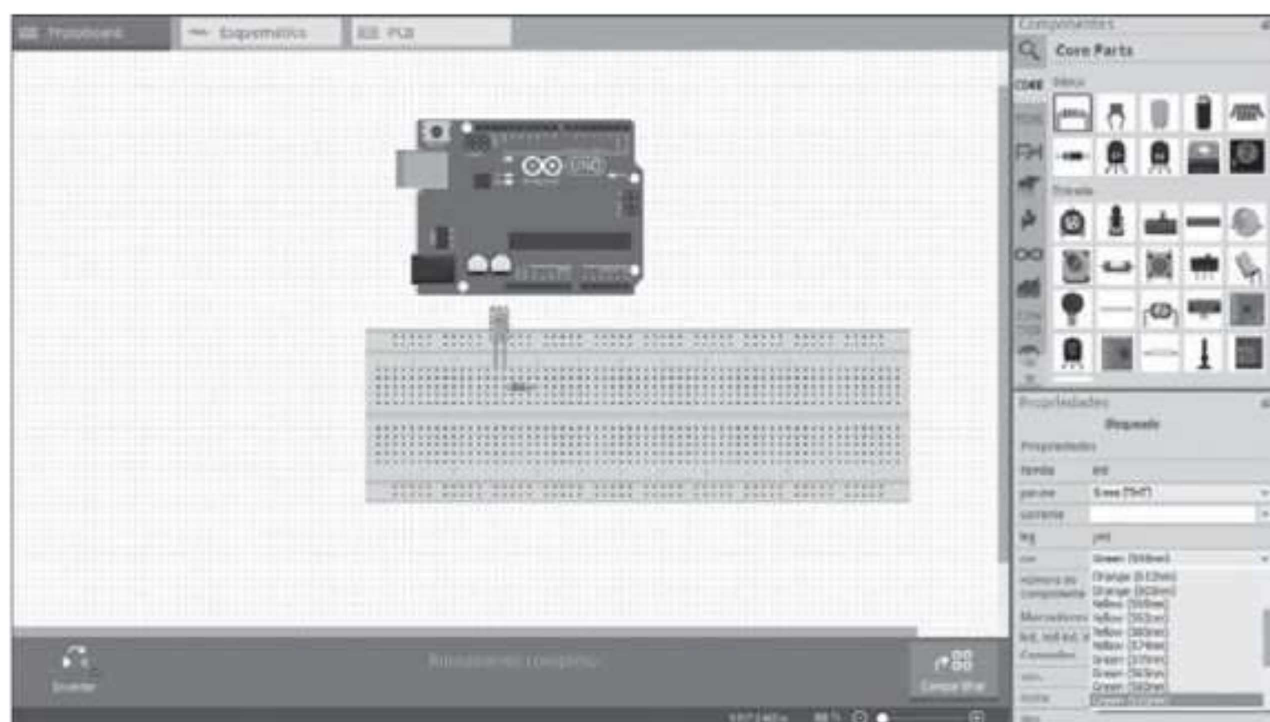


Figura 13.6 – Alterando a especificação (propriedade) de um componente.

Após posicionarmos todos os componentes na área de trabalho e realizarmos as configurações necessárias nas propriedades daqueles utilizados, devemos realizar a conexão desses por meio da utilização de jumper cables (fios). No Fritizing as conexões são feitas clicando em um ponto de conexão, ou seja, uma conexão (furo) da protoboard, pino do Arduino ou terminal (perna) de algum componente, entre outros. Dessa forma, devemos clicar em um ponto de conexão e arrastar o mouse até o outro ponto que será conectado.

Na Figura 13.7 podemos notar o resultado ao clicarmos na linha de conexão da protoboard junto ao resistor e arrastar o mouse até o pino 13 do Arduino. Note também que a cor do jumper cable pode ser alterada na paleta Propriedade.

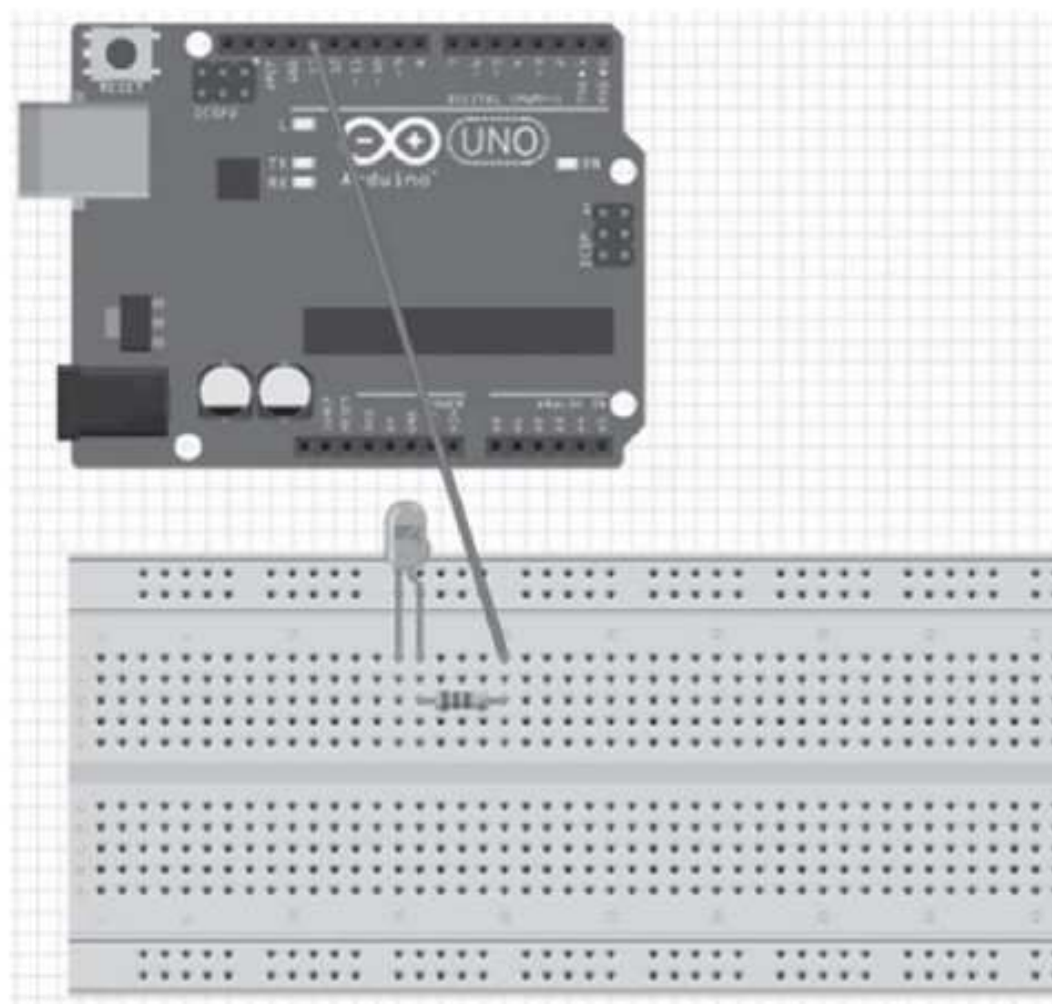


Figura 13.7 – Realizando uma conexão com jumper cable (fio).

Um jumper cable pode ser moldado, clicando em algum ponto dele. Ao clicar e arrastar, você verá a formação de um ponto de flexão (dobra). Em um jumper cable é possível criar inúmeros pontos de flexão, ou seja, quantos forem necessários para uma melhor apresentação do esquema.

Após completar a montagem, podemos exportar o diagrama criado para um dos diversos formatos de imagem suportados pelo Fritzing. Para fazer isso, acesse o menu "Arquivo", escolha "Exportar", depois "como Imagem" e selecione o formato de imagem desejado (Figura 13.8).

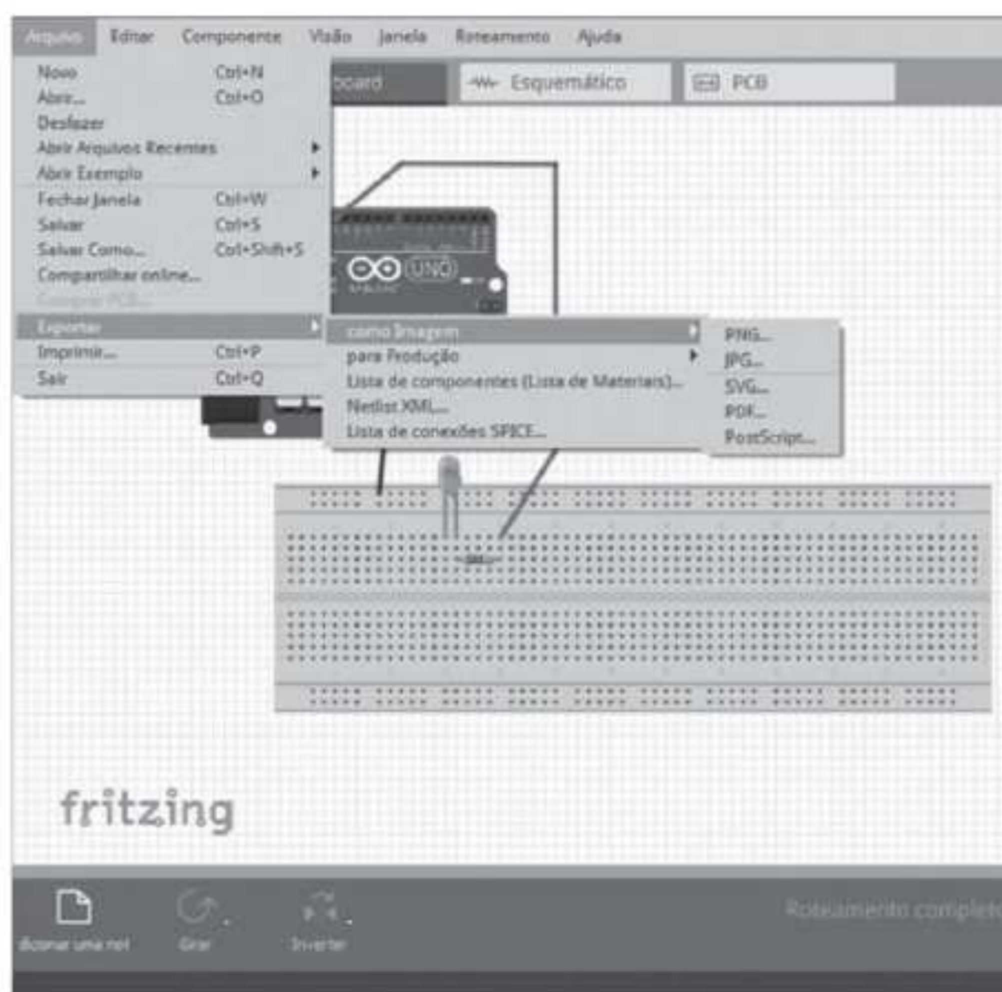


Figura 13.8 – Exportando um diagrama.

Exercícios

1. Reproduza no Fritzing o projeto mostrado na Figura 13.9, considerando os seguintes componentes: 1 Arduino, 1 resistor de 220 ohms (vermelho, vermelho, marrom), 1 resistor de 10 kohms (marrom, preto, laranja), 1 LED vermelho, 1 LDR, 1 protoboard e jumper cables para realizar as conexões.

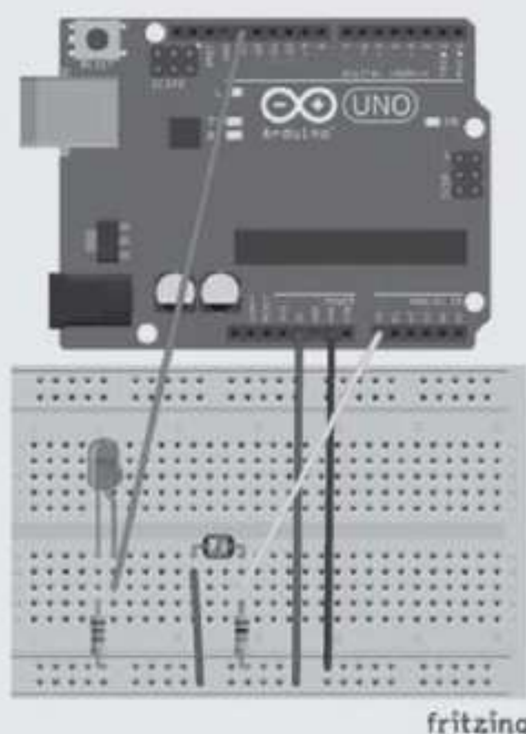


Figura 13.9 – Uso do LDR.

2. Reproduza no Fritzing o projeto mostrado na Figura 13.10, considerando os seguintes componentes: 1 Arduino, 1 Real Time Clock (RTC DS-1307), 1 display de LCD 16x2, 1 resistor de 330 ohms (laranja, laranja, marrom), 1 potenciômetro de 10 kohms, 1 protoboard e jumper cables para as conexões.

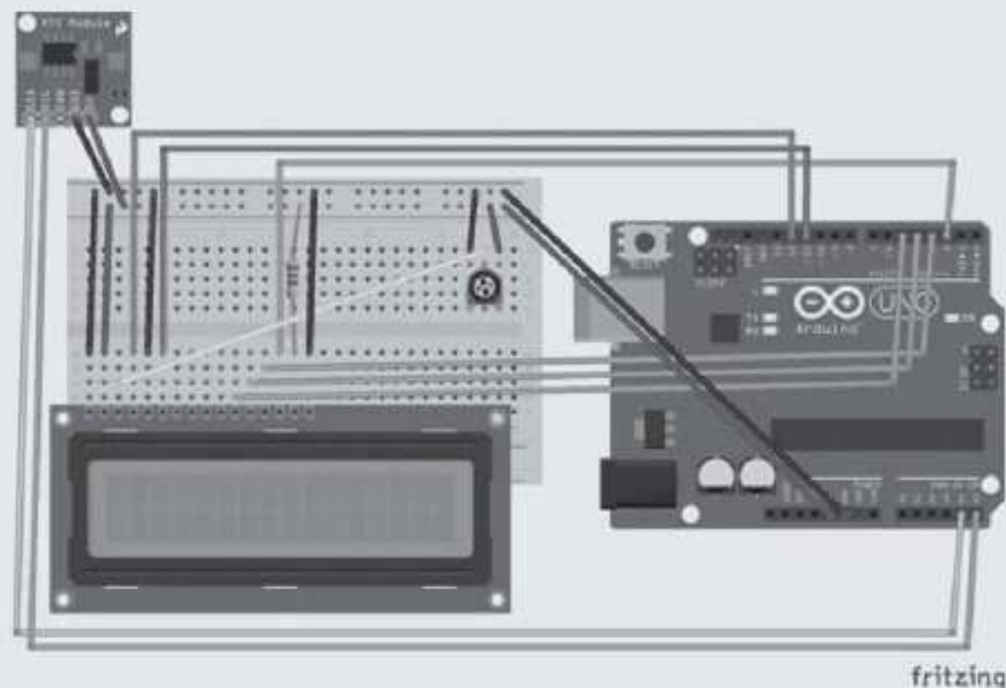


Figura 13.10 – Relógio com display de LCD.

Bibliografia

ARDUINO. Disponível em: <http://www.arduino.cc>. Acesso em: 2 dez. 2014.

BANZI, M. **Primeiros passos com o Arduino.** São Paulo: Novatec Editora, 2011.

FRITZING. Disponível em: <http://www.fritzing.org>. Acesso em: 2 dez. 2014.

McROBERTS, M. **Arduino básico.** São Paulo: Novatec Editora, 2011.

PROCESSING. Disponível em: <http://processing.org>. Acesso em: 2 dez. 2014.

A

Outros Modelos do Arduino

Os projetos apresentados neste livro foram desenvolvidos utilizando o Arduino Uno R3, o modelo mais popular e mais utilizado atualmente. Porém, outros modelos podem ser usados, sem que haja necessidade de alterar os circuitos eletrônicos empregados e a programação dos *sketches*. Neste apêndice serão apresentados os três modelos mais populares, depois do Arduino Uno R3, que são: o Arduino Mega 2560, o Arduino Nano e o Arduino Leonardo.

Arduino Mega 2560

O Arduino Mega 2560 é perfeito para projetos que apresentam maior grau de complexidade e que, por consequência, exigem maior número de entradas e de saídas que o Arduino Uno R3.

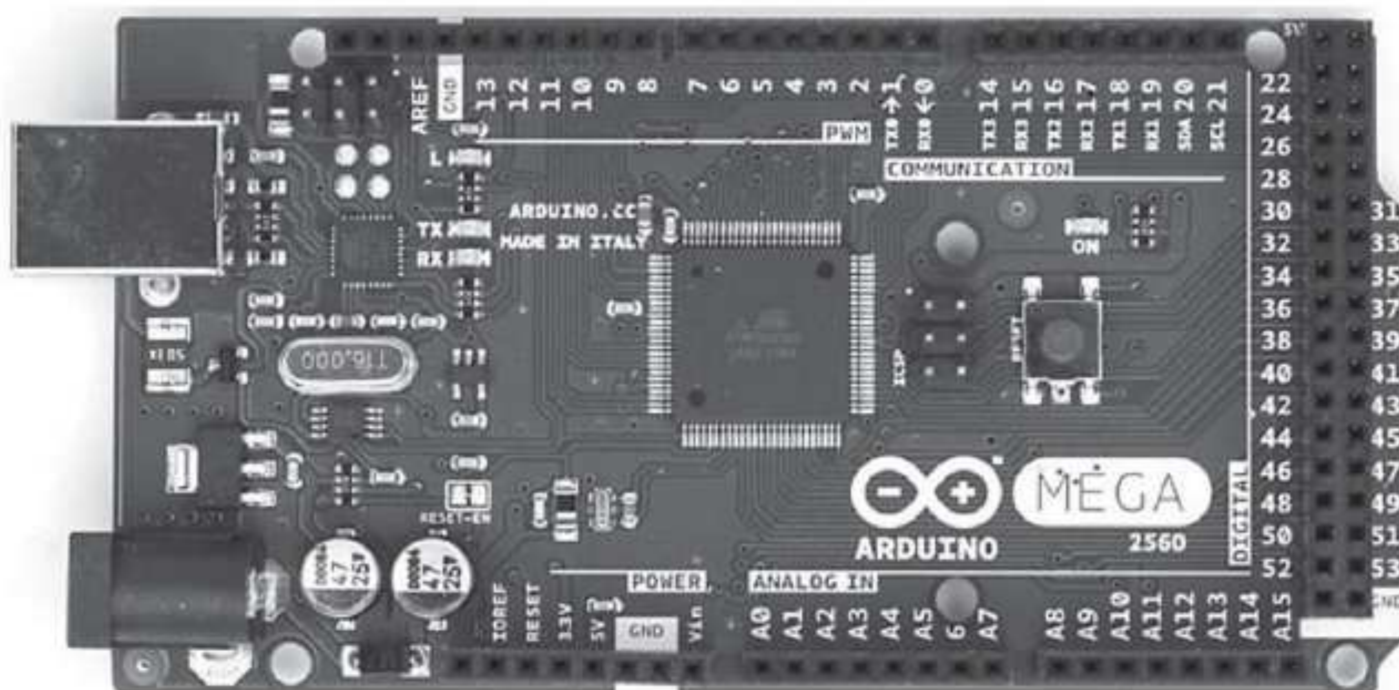


Figura A.1 – Arduino Mega 2560.

Na Figura A.1, a placa apresenta 54 portas digitais – destas, 15 podem ser usadas como PWM (*Pulse Width Modulation*) – e 16 portas para entradas analógicas.

Tabela A.1 – Especificações da placa

Especificações	Arduino Mega 2560
Microcontrolador	ATmega2560
Tensão de operação	5V
Pinos de I/O digitais	54 (14 com saída PWM)
Pinos analógicos	16
Memória flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Velocidade do clock	16 MHz

Arduino Nano

O Arduino Nano é uma placa que pode ser diretamente conectada à *protoboard*. Suas dimensões são bastante reduzidas quando comparada a outros modelos. Porém, apresenta a mesma quantidade de pinos do Arduino Uno R3.

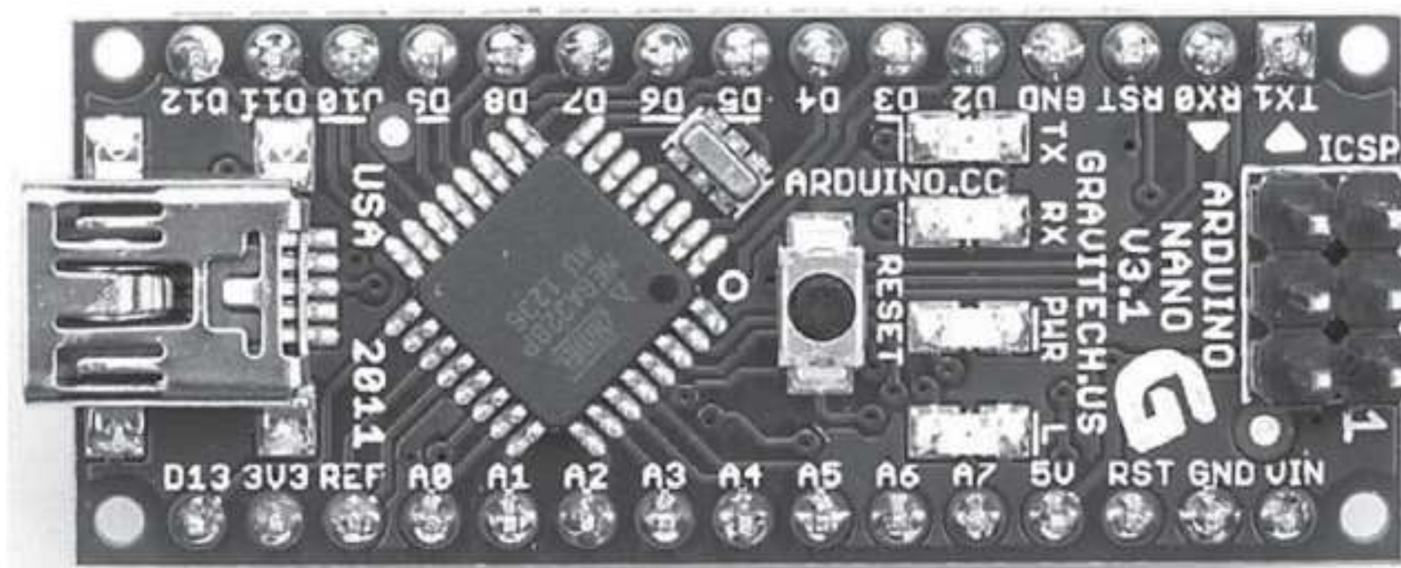


Figura A.2 – Arduino Nano.

A seguir, são mostradas as principais especificações do Arduino Nano, que pode ser encontrado ao utilizar o microcontrolador ATmega168 ou ATmega328.

Tabela A.2 – Especificações da placa

Especificações	Arduino Nano
Microcontrolador	ATmega168 ou ATmega328
Tensão de operação	5V
Pinos de I/O digitais	14 (6 com saída PWM)
Pinos analógicos	8
Memória flash	16 KB (ATmega168) ou 32 KB (ATmega328)
SRAM	1 KB (ATmega168) ou 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) ou 1 KB (ATmega328)
Velocidade do clock	16 MHz

Arduino Leonardo

O Arduino Leonardo diferencia-se dos demais modelos principalmente por seu microcontrolador (ATmega32U4), que é mais poderoso. Visualmente, a placa (Figura A.3) é bastante similar ao modelo Uno R3, em razão de suas dimensões e configuração; contudo, a presença do conector USB do tipo micro, ao invés do regular, é a principal diferença física.

Essa placa é considerada a "evolução do Arduino" por conta de seu maior poder de processamento e de algumas outras funcionalidades, como: integração da comunicação USB, sendo tratada no microcontrolador principal (demais modelos possuem microcontrolador específico para isso); e possibilidade de conectar periféricos como teclados, mouses ou *joysticks* diretamente na placa.

Tabela A.3 – Especificações da placa

Especificações	Arduino Leonardo
Microcontrolador	ATmega32U4
Tensão de operação	5V
Pinos de I/O digitais	20 (7 com saída PWM)
Pinos analógicos	12
Memória flash	32 KB
SRAM	2,5 KB
EEPROM	1 KB
Velocidade do clock	16 MHz